A Practical Guide to CP/M®

Carl Townsend

679035-60
the electronics boutique
Games 'n' Gadgets
.99 ~~$1.99~~

Practical Guide to CP/M

Practical Guide to CP/M

Carl Townsend

dilithium Press
Beaverton, Oregon

© 1983 by dilithium Press. All rights reserved.

No part of this book may be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system without permission in writing from the publisher, with the following two exceptions: any material may be copied or transcribed for the nonprofit use of the purchaser, and material (not to exceed 300 words and one figure) may be quoted in published reviews of this book.

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data

Townsend, Carl, 1938-
Practical guide to CP/M.

Bibliography: p.
Includes index.
1. CP/M (Computer operating system) I. Title.
II. Title: Practical guide to C.P./M.
QA76.6.T68 1983 001.64 83-7251
ISBN 0-88056-077-0

CP/M is a registered trademark of Digital Research of Pacific Grove, California.

Printed in the United States of America

dilithium Press
8285 S.W. Nimbus
Suite 151
Beaverton, Oregon 97005

Trademark Acknowledgments

CompuPro	Godbout Electronics
CompuServe	Consumer Information Service, An H & R Block Company
CP/M, MP/M, TEX	Digital Research
dBase II	Ashton Tate
8080, 8008, 8086 chips	Intel Corporation
IBM Selectric	International Business Machines Corporation
KBASIC	EIDOS Systems Corporation
Microsoft BASIC	Microsoft Corporation
Osborne I	Osborne Computer Company
PDP-10	Digital Equipment Corporation
The Source	Source Telecomputing Corporation
SELECT	Select Information Systems, Inc.
SuperCalc	Sorcim Corporation
UNIX	Bell Laboratories
WordStar	MicroPro International Corporation
Z80 chip	Zilog Corporation

Acknowledgements

Grateful appreciation is expressed to the many microcomputer pioneers who made this book possible. A special thanks to Ninad Freedman for allowing us to "beta test" his SELECT word processor and to use SELECT to prepare the manuscript for this book. Thanks to Tim Pugh and Ward Christensen and many others who have contributed to the user groups. Thanks to Tom Jorgenson and others in both The Source and CompuServe for their contributions. Very special thanks to Gary Kildall, the creator of CP/M.

Table of Contents

Introduction			1
Chapter 1	**A History and Overview of CP/M**		3
	1.1	Beginnings	3
	1.2	The First CP/M	4
	1.3	The Growth of CP/M	6
	1.4	The CP/M System	7
	1.5	CP/M: A Software Bus?	9
Chapter 2	**Saving Your Programs**		11
	2.1	Logical and Physical Information	11
	2.2	The CP/M Directory	12
	2.3	The Single-Density Disk	16
	2.4	The Directory Entry	17
	2.5	Allocation Strategy	18
	2.6	Bootstrapping CP/M	19
Chapter 3	**Interfacing to the IOBYTE**		21
	3.1	What's the IOBYTE?	21
	3.2	The Functions of the IOBYTE	24
	3.3	Implementing the IOBYTE	26
	3.4	Switching the IOBYTE–Using STAT	28
Chapter 4	**Using PIP and SUBMIT**		33
	4.1	Two Modes	33
	4.2	PIP with Device Names	35
	4.3	Special Transfers and Parameters	36
	4.4	PIP Enhancements in CP/M 2.2 and MP/M	37

	4.5	Using SUBMIT	38
	4.6	Special Tricks Using SUBMIT	40
Chapter 5	**Talking to Other Devices**		43
	5.1	Program Controlled I/O	43
	5.2	Interrupt Controlled I/O	45
	5.3	DMA Controlled I/O	45
	5.4	Application Areas	47
Chapter 6	**CP/M I/O Function Calls**		51
	6.1	I/O Interfacing for Consoles and Printers	51
	6.2	BDOS Calls	52
	6.3	Direct I/O	56
	6.4	BDOS Versus Direct Calls	57
	6.5	Disk Calls	58
Chapter 7	**Creating CP/M Systems**		67
	7.1	Types of System Generation	67
	7.2	Copying CP/M	68
	7.3	Changing an Existing CP/M System	69
	7.4	Offset and Bias	72
	7.5	The Creation	74
	7.6	Calculating the Offset and Bias	76
	7.7	Assembling the BIOS and BOOT	78
	7.8	The Generation	80
Chapter 8	**Database Management**		83
	8.1	Database Management	84
	8.2	KBASIC and Keyed Indexed Sequential Indexing	86
	8.3	A KBASIC Example	89
	8.4	The Phonic Function	94
	8.5	KBASIC Speed	95
	8.6	dBase II	95
	8.7	Looking to the Future	99
Chapter 9	**Recovering Lost Programs**		101
	9.1	Recovering Programs Lost During Execution	104
	9.2	Random Number Generation	105
	9.3	Automatic Startup	106

Chapter 10 Multi-User and Networking Systems 109
10.1 MP/M Components 110
10.2 Bank Select 112
10.3 Consoles and Users 113
10.4 MP/M Operation 114
10.5 MP/M Commands 115
10.6 Generating MP/M Systems 116
10.7 Distributed Processing 118

Chapter 11 Future Developments 119
11.1 Database Management 119
11.2 Networking 119
11.3 Concurrency 120
11.4 Conclusion 120

Appendix A Memory Maps 121

Appendix B Command Tables 125

Appendix C Hardware Suppliers 133

Appendix D Software Suppliers 137

Appendix E CP/M Diagnostics 153

Appendix F CP/M User Groups 157

Appendix G Operational Procedures 163

Appendix H ASCII Codes 165

Appendix I Sample Assembly Listing 185

Glossary 211

Bibliography 219

Index 223

Introduction

To many people, CP/M is a forbidding mystery. The user manuals seem difficult or impossible. Everybody seems to have a CP/M system; yet disks from one system will not run on a competitor's system. To the user CP/M seems purposely loaded with unintelligible messages such as "BDOS ERROR" or "R/O." Mazes of undecipherable code can trap the unwary in the BIOS module. Software is often marketed by non-professionals whose companies disappear overnight. Software quality is difficult for the user to evaluate. Even the best software is often released with hidden "bugs" that may or may not be resolved by the time the reviews start appearing. Interfacing hardware and software is difficult for someone without computer experience.

Even with these difficulties, however, CP/M is a valuable microcomputer tool. It is the most revolutionary operating system marketed in the last decade and has brought computer power within financial reach of the small businessman and hobbyist. Over 400,000 people have CP/M and run it on hardware from hundreds of manufacturers—Xerox, IBM, Wang, Lanier, and Hewlett Packard, to name just a few. CP/M usage triples each year. There is more software available for CP/M than for any other operating system and much of this software is both well-developed and inexpensive.

This book is intended for the CP/M user who has a few months of experience and for the serious CP/M user who wants to learn more about the operating system. It is not intended for a beginner. Several books are already available for the beginner, including *How to Get Started With CP/M* by

the author. The bibliography in the back of the book lists several books and magazine articles available on CP/M. This book should be used with the CP/M manuals from Digital Research and at least two or three from the bibliography. Prepare yourself well for the journey into CP/M. It will be work, but well worth the effort!

Chapter 1

A History and Overview

1.1 BEGINNINGS

In 1972 a man named Gary Kildall had just finished his doctorate in Computer Science and was an instructor at the Navy Postgraduate School in Monterey, California. Looking for a challenge, he consulted with Intel in his spare time on a variety of projects. His small company, Microcomputer Application Associates, was the beginning of what was to become Digital Research. Intel was already a large corporation, but the microcomputer division was a tiny department that had a big vision. Gary Kildall saw his first 8080 chip in 1973 as a jumble of wires and silicon in Masatushi Shima's laboratory. Few people at the time realized the impact this tiny mass of silicon would have on technology.

Kildall proposed to develop a high-level language called PL/M (or Programming Language for Microcomputers) for the tiny chip that would, for the most part, replace the need for assembly language programming. PL/M was defined as a subset of PL/1 with some extras. PL/1, in turn, had been developed by IBM in 1964 as a combination of ALGOL, FORTRAN, and COBOL, along with a few new additions. IBM intended it to be an all purpose language that would work in both business applications (such as COBOL environments) and scientific applications (such as FORTRAN environments).

In 1973 there was no 8080 system on which to develop PL/M. Gary Kildall proposed to develop the compiler in FORTRAN for operation on a PDP-10 time-shared computer.

Once this was working, Kildall planned as a second phase to write a PL/M compiler that could compile itself into the 8008 and 8080 languages. Before the end of 1973 he had completed the first part of this ambitious project.

The problems started when he began working with the small computer system. There were no floppies in 1973. Everyone used paper tape and a noisy, clattering device called the Model 33 Teletype. Paper tapes were loaded, read, and often reloaded several times until a successful load was achieved. From an economical standpoint, such a small computer system could never replace the time-shared minicomputer.

Down the road a few miles from Intel there was a small company called Shugart Associates that was beginning to make a floppy disk drive. Kildall persuaded Dave Scott, who was then marketing manager at Shugart, to donate one of their worn-out drives (used for 10,000 hours) to the cause. Scott also threw in a bearing repair kit. This hardly helped at the time. Without hardware experience and using the naked, worn-out disk drive with no controller, power supply, or cables Kildall turned again to the PDP-10 and software simulation.

1.2 THE FIRST CP/M

Kildall neded some type of operating system for the microcomputer to support and test the PL/M compiler. The first CP/M was designed to do this and was completed on the PDP-10 almost a year before it ever ran on the 8080. The operation of CP/M was *simulated.* No microcomputer system existed for it in 1973.

In the fall of 1973 John Torode began working with Kildall. Torode completed Kildall's early controller development and got the microcomputer system going. CP/M ran the first time it was tried and printed a prompt on the Teletype. A few tests confirmed they now had the power of a computer at their fingertips; Kildall and Torode retired for the evening to celebrate!

CP/M was designed with reliability and simplicity as the key criteria. Unlike most operating systems available at that time, CP/M did not use the linked directory concept. Abandoning the linked directory concept speeded up directory

access and updating on the large disk systems used in minicomputers and mainframes. It also avoided another major problem. In a linked directory system, if the linkage is lost from any error in the directory, it is almost impossible to use or reconstruct the entire directory. In CP/M there were no linked directory lists and the directory could be recovered except in very unusual cases. Disks could be changed without losing files or mixing data records. And even with serious system crashes, files were almost never lost.

Intel took very little interest in Kildall's new operating system, but it did use the PL/M. Intel could not visualize the need for a single-user operating system until 1976 and chose instead to continue the time-shared development. Kildall continued on his own developing utilities and getting the system going on four different controllers.

John Torode resigned from Microcomputer Applications Associates and developed a complete microcomputer system which was marketed under his company name of Digial Systems, which later became Digital Microsystems. The first commercial license of CP/M was in 1975 with contracts between digital Systems and Omron of America to use the system at Lawrence Livermore Laboratories as part of an intelligent terminal to monitor programs in the Octopus network.

In 1976 Kildall was approached by Glen Ewing who was consulting with IMSAI. IMSAI had been shipping a large number of disks with the promise of an eventual operating system, but no operating system existed. Could Kildall adapt CP/M to work with IMSAI's controller?

Out of Kildall's dialogue with Glen Ewing an idea evolved. The hardware-dependent portions of CP/M were separated out to a single assembly-language module called the BIOS. The portions that were not hardware dependent remained in PL/M. In this way any manufacturer could easily modify CP/M to work with their particular hardware. Thus CP/M became a mixture of hardware-independent PL/M code and the hardware-dependent assembly-language BIOS module. The result was CP/M 1.3 which was sold by IMSAI as IMDOS. In 1976 Gary Kildall, with his wife Dorothy, started Digital Research to market CP/M and continue the development.

The microcomputer revolution had begun. With the introduction of the new microcomputer systems, computing power

was inexpensive. Computer "junkies" and hobbyists created a whole new market overnight. Amidst fears of being ripped off and with the affirmation of computer guru, Jim Warren, Kildall released CP/M 1.3 to the public for $70. The rest is history. Other companies soon realized they could not afford the investment to develop an operating system and compete with CP/M. As the volume of application software grew rapidly, it soon became necessary for manufacturers to provide some type of CP/M for their own system, even if they did have their own operating system.

1.3 THE GROWTH OF CP/M

CP/M has evolved through the years and is now an advanced-design system that is the product of many creative minds at Digital Research. In 1979 CP/M 2.0 was released with a table-driven disk driver. On a disk system, after a sector is read there is a short delay for the operating system to digest the information. By the time the system is ready to read the next sector, the disk has rotated several sectors. The system had to wait until the disk rotated back to the proper position before reading the next sector. To resolve this and to speed up the disk access, early CP/M versions (1.4) read the second sector as six sectors after the first. Thus, physical sector seven was logical sector two, and physical sector thirteen was logical sector three. This six-sector offset (or skew factor) was fixed. CP/M 2.0, being table driven, had the skew set in the BIOS and controlled by the disk manufacturer. In addition, the table could be set up for any number of tracks and sectors on the disk. CP/M could now be optimized for minidisks, hard disks, or full-size floppies of various configurations. This had a price, however, as disks on one system are often unreadable by another system.

As CP/M became widely accepted, Digital Research realized they needed an upgraded operating system for multiuser operation and real-time processing. They developed the idea of taking the pioneering concepts of CP/M and applying them to a multi-user system. These two concepts included the non-linked directory and the separation of the hardware-dependent portions from the proprietary portion written in PL/M.

In 1980 the first version of MP/M was released, meeting these objectives. A real-time nucleus supported sequential

program processes, including a CP/M-compatible file manager. In 1981 a major upgrade of MP/M increased file protection.

The founders of Digital Research also realized a simple microprocessor had limited processing that was often too slow for many business environments with multi-user applications. The processor portion is relatively inexpensive, so in late 1980 Digital Research released their first version of CP/NET, a multiprocessor "shell" designed to interconnect an MP/M system with one or more CP/M systems.

As the state of the art in hardware continued to advance, manufacturers began to build more complex microprocessors. Responding to this need, in 1980 Digital Research introduced CP/M-86 to extend CP/M to the 16 bit 8086 and 8088 microprocessors manufactured by Intel.

CP/M, which started as a support operating system to test a language on a development system, has now emerged as a formidable family of products with over 400,000 users.

1.4 THE CP/M SYSTEM

CP/M, as sold by Digital Research, has a machine-language proprietary portion that is independent of the system hardware (the invariant portion) and a hardware-dependent BIOS portion that is machine dependent (the variant portion). In the Digital Research version, the variant portion is for the Intel Intellec system. Individual system manufacturers license CP/M from Digital Research and rewrite the BIOS portion for their own hardware. The rewritten CP/M is then sold to the user. The minimum hardware requirements for a CP/M 2.0 system includes an 8080 or Z80 processor, 20K of memory, a floppy or hard disk drive with controller, and a video terminal.

The CP/M operating system is generated by the manufacturer (or user) to load at the highest portion of the computer's memory. For a single-user system, the maximum memory size is 64K. If you start with a smaller memory size and later expand your memory, CP/M must be reconfigured before it can use the extra memory you added (see Chapter 7). CP/M also uses 256 bytes of memory at the low end of memory. With the 8080, 256 bytes is defined as one page, so CP/M uses one page at the low end of the memory.

To the user, CP/M appears to have two parts. One part is proprietary and independent of the hardware used. It is purchased and licensed from Digital Research. The second part, a nonproprietary part that varies with the hardware, is called the BIOS. The user will periodically alter the BIOS as various peripherals are added or changed. The BIOS contains the physical I/O device drivers.

Technically, CP/M is divided into three components. The proprietary part is considered to be two components. The first is called the BDOS; the second the CCP. The three components can be defined as:

BIOS (Basic Input/Output System) Basic input and output routines. This module is hardware dependent and a source listing is generally supplied with the system. It is altered when peripherals are added or changed. The BIOS is generally proprietary to the system manufacturer.

BDOS (Basic Disk Operating System) Disk manager and software for dynamic file allocation. Hardware independent and proprietary to Digital Research.

CCP (Console Command Processor) Reads and interprets commands from the console. Hardware independent and proprietary to Digital Research.

The BIOS always loads at the highest part of the computer memory and the BDOS below this (Figure 1.1). The CCP loads below the BDOS and can be overlaid by user programs. The BIOS and BDOS together are often referred to as a single module called the FDOS (*Functional Disk Operating System*). All of the memory from 100H to the start of the CCP is not used by CP/M and is available to the user. It is called the *Transient Program Area* or TPA.

All CP/M systems load some variation of this configuration; however the number of bytes allocated for the BIOS will vary between systems. The CP/M version supplied by Digital Research allocates only two pages (200H) for the BIOS. Manufacturers, however, can use a larger BIOS if the remaining portion of CP/M is moved downward in memory to compensate.

Application programs generally use the BDOS and BIOS; but not the CCP. The CCP is normally overlaid by the ap-

plication program and reloaded with CP/M at the program completion. This is called a *warm boot* (see Chapter 7). Locations 002 and 003 normally contain the address of the first location of FDOS, the portion (BDOS and BIOS) that is not overlaid. Application programs, therefore, can read these locations to determine how much memory they can use. Long programs may overlay the BDOS and make direct calls to the BIOS (see Chapter 6), but this is a dangerous program practice in which the application program can lose compatibility between CP/M systems.

To add further confusion, many manufacturers use PROMs in portions of the 64K of memory that CP/M uses. To use CP/M with these systems, the CP/M is relocated and has a different "virtual" memory image. These systems include:

Radio Shack–TRS-80 Models II and III
Heathkit–H8 and H89 (and Zenith Z89)

1.5 CP/M: A SOFTWARE BUS?

CP/M, one competitor claims, does not exist because there are so many versions and near zero compatibility. This manufacturer was confusing CP/M as a concept (the PL/M invariant portion and interface requirements for the BDOS and disk organization) and CP/M as an operating system. the industry is already referring to the BDOS interface definition (the invariant portion) and directory concept as a *software bus*. This part does not vary between manufacturers. The memory image of CP/M varies, the disk size and sectoring, and the method of booting and generating systems vary. For this reason CP/M disks are often not compatible between systems. The BDOS interface, disk, and file concept, however, are always the same.

Table 1.1. CP/M Time Line

1972	Gary Kildall finishes his Doctorate in Computer Science and begins consulting at Intel to develop PL/M.
1973	Laboratory version of the 8080 created. PL/M developed on PDP-10. CP/M simulated on PDP-10.
1974	John Torode develops floppy disk hardware controller and CP/M runs on the 8080 for the first time.

1975	First commercial license granted.
1976	Hardware-dependent portions of CP/M separated out and first CP/M 1.3 sold through IMSAI. CP/M released to public. Digital Research is founded.
1979	CP/M 2.0 released with table-driven file manager.
1980	MP/M released. CP/Net released. Sixteen-bit CP/M-86 released.
1981	MP/M II released.
1982	CP/M 3.0 released.

*All memory addresses (e.g., 200H) are in hexadecimal notation.

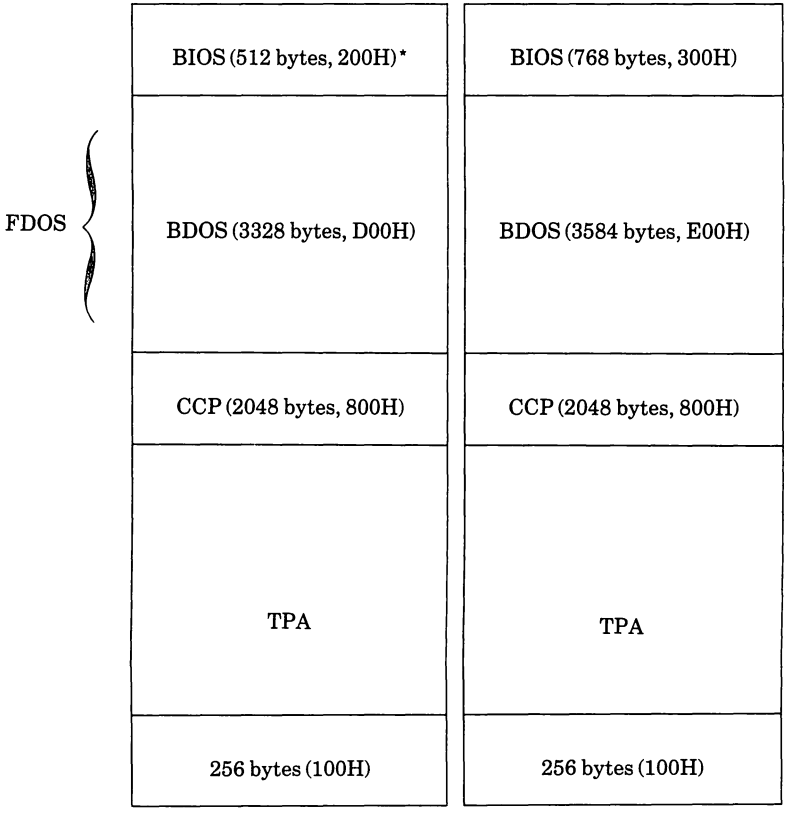


Figure 1.1. CP/M Memory Maps

Chapter 2

Saving Your Programs

One of the features of CP/M that has contributed most to its success is the method it uses to process and save files. Floppy disks or hard disks are used as the primary mass storage unit for CP/M systems, and a large number of manufacturers provide a variety of eight-inch disk drives, mini-disk drives, and hard disk drives. Although there is considerable hardware variation between manufacturers, CP/M–and its multi-user cousin MP/M–interface easily with all of these. How this works and the pioneering CP/M concept of the nonlinked directory are the subjects of this chapter.

2.1 LOGICAL AND PHYSICAL INFORMATION

For your convenience as a user, information on a disk is logically arranged in volumes, files, records, and characters (Figure 2.1). A typical word processing program, for example, might store a line of characters as a single *record* and separate records by carriage return/line feed combinations. A related group of records, in turn, forms a single *file*–for example, a chapter of a book. A group of related files together form a *volume*, which might be a single disk in a collection of disks. This logical organization of information may differ between programs, as another word processor might separate records by a different method. Records may be of varying length or all the same length. Thus, there are different logical information divisions, but each is designed for your convenience as a user.

Physically, however, CP/M can only store information in *sectors*. On a single-density floppy disk with IBM format a sector consists of 128 bytes (characters). Each time CP/M does a write or a read on this system it always writes or reads 128 bytes. On this type of disk 26 sectors form a *track*, and there are 77 tracks arranged concentrically on the disk.

Thus information is *physically* divided into bytes, sectors, tracks, and disks. Logically, the information is in characters, records, files, and volumes. The mapping between the logical and physical organizations of the data is stored in a *directory* on the third track of the disk (the first track is track 0 so the directory is actually on track 2).

Suppose, for example, you are using the CP/M editor and load three lines of text into the computer using the "3A" command. This will probably mean reading more than one sector. As each sector is read, the 128 bytes are stored in a buffer area (more on this later). The editor then pulls the data out of this buffer and loads it into the program buffer in the TPA. The next line to be read has probably already been read from the disk into the CP/M buffer. Thus, if you do an "A", you may not even hear the disk head load. Subsequent "A" commands will not cause a read from the disk again until the 128-byte buffer is empty.

Starting with CP/M version 2.0, the number of bytes read at a time can be set when the system is created. With hard disks, for example, the system can be configured to read 256 or 512 bytes at a time.

2.2 THE CP/M DIRECTORY

CP/M further divides information logically into files, extents, groups (or allocations), and logical sectors (Figure 2.2). This may sound confusing at first, but it is essential to understanding the CP/M information management concept. You the user think in terms of files, records, and characters; CP/M uses files, extents, groups, and logical sectors; the hardware uses tracks, sectors, and bytes (Figure 2.1).

Like the other methods of organizing data, this directory organization is simply a convenient way of grouping smaller units of data into larger units. A *logical sector* is the smallest

unit, although the size of a logical sector varies from system to system. A collection of related logical sectors is called a *group*; a collection of related groups is called an *extent*; and, finally, a group of extents form a *file*.

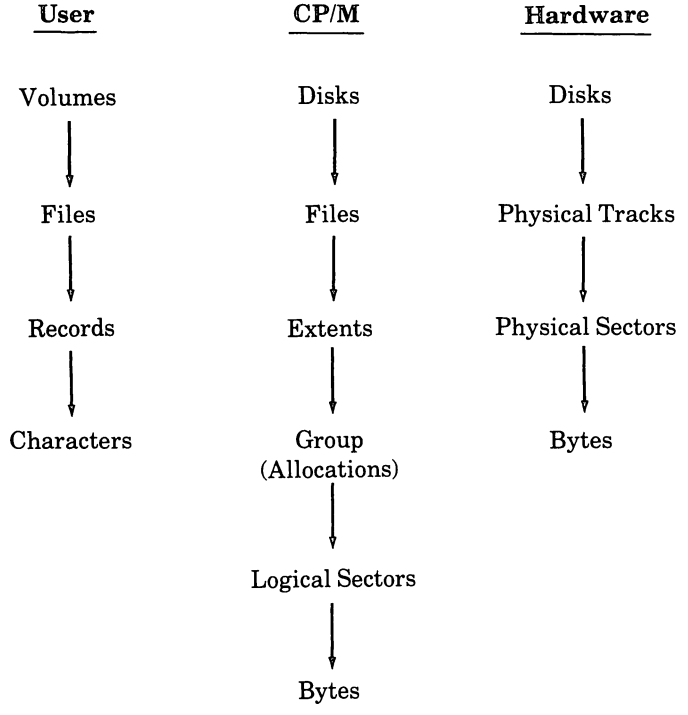


Figure 2.1. Logical and Physical Information Division

The smallest unit a directory can address is one group. As a file grows, it is assigned additional groups as long as they are available on the disk. After each 16 groups, a new extent is opened creating an additional directory entry. If you do a STAT on a CP/M 1.4 disk, the smallest file size you will ever see is 1K—a single group. On CP/M 1.4, 128 bytes are read at a time, but the directory allocates 1K each time the program or file needs more space. On a hard disk with 1K logical sectors the smallest file size you will ever see is 8K.

CP/M can assign as many as 16 extents to a single file. Some CP/M books and articles refer to a logical sector as a

record. However, this may be easily confused with our previous concept of records, and we avoid this usage.

Figure 2.3 shows part of a CP/M directory. Notice that some files have more than one disk entry. With CP/M 1.4, each entry can address 16,384 bytes of a file (the usual computer terminology for this is 16K bytes):

$$\frac{128 \text{ characters}}{\text{sector}} \times \frac{8 \text{ sectors}}{\text{group}} \times \frac{16 \text{ groups}}{\text{extent}} = 16384 \frac{\text{characters}}{\text{extent}}$$

Thus, as a file grows beyond 16K, the user program must create additional extents and directory entries using CP/M routines (more on this later). There can be up to 64 directory entries on a single disk with CP/M 1.4.

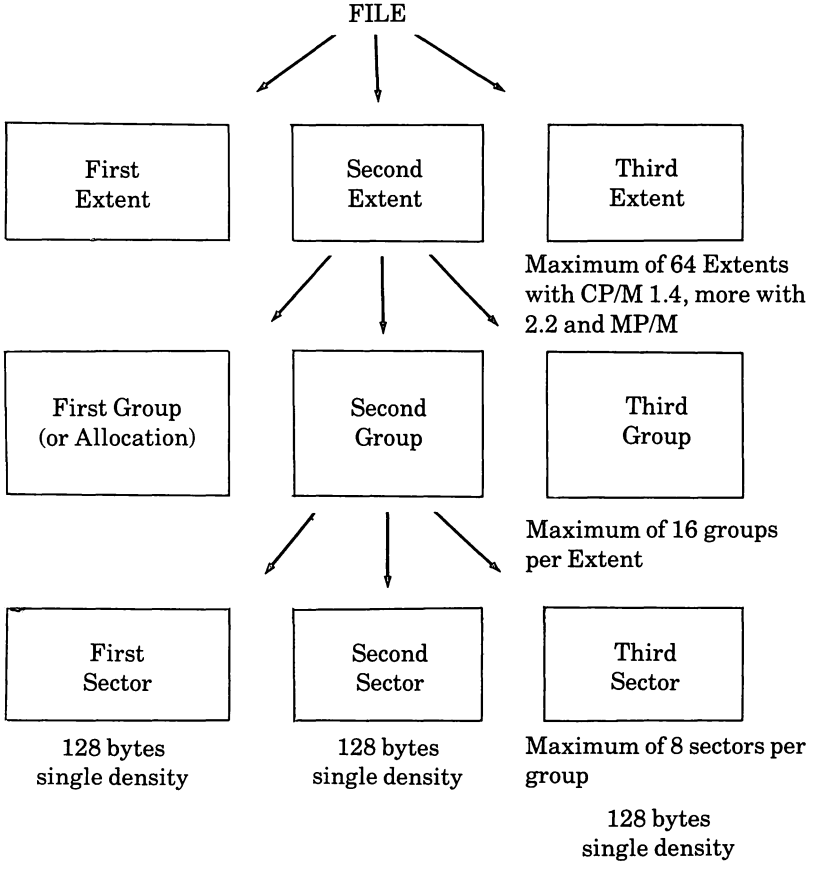


Figure 2.2. CP/M Directory Division

Since the extent, group, and logical sector are all logical concepts, the amount of storage represented by each can vary with CP/M 2 and MP/M systems. For example, a 32-megabyte hard disk in an MP/M system may use groups of 8K characters and read logical sectors of 1K (1024) bytes at a time. A double-density floppy disk may use logical sectors of 256 bytes each. The logical-to-physical mapping for a particular CP/M system is stored in the BIOS module and established by the system manufacturers. In a CP/M 1.4 system it is fixed and cannot be altered.

Another important concept used in logical sectoring is the *sector skew*. In Figure 2.3 you will notice the first part of the directory entries are read beginning at group 0, track 2, (logical) sector 1, and physical sector 1. The next part is read at group 0, track 2, (logical) sector 2, and physical sector 7. After CP/M reads the first sector, it requires a few milliseconds for the information to be processed. When CP/M returns for the next immediate read, physical sector 7 is under the disk head. Rather than wait for another disk rotation, CP/M assumes sector 7 is the second logical sector and reads it as logical sector 2. Thus, while physical sectors are contiguous (end-to-end) on the disk, the logical sectors are not. This offset, or *sector skew*, is set at 6 sectors in CP/M 1.4. In CP/M 2.X or MP/M the amount of skew is set in the BIOS.

```
:g0
G=00:0, T=02, S=01, PS=01*

:d
00  00544558  58202020  20434F4D  00000080   *.TEXX   COM.  .  .  .  *
10  02030405  06070809  0A0B0C0D  0E0F1011   *. . . . . . . . . . . . . . . . *
20  00544558  58202020  20434F4D  01000080   *.TEXX   COM.  .  .  .  *
30  12131415  16171819  1A1B1C1D  1E1F2021   *. . . . . . . . . . . . . . . ! *
40  00544558  58202020  20434F4D  0200000B   *.TEXX   COM.  .  .  .  *
50  22230000  00000000  00000000  00000000   *"#. . . . . . . . . . . . . . . *
60  00442020  20202020  20434F4D  0000000A   *.D      COM.  .  .  .  .  *
70  24B40000  00000000  00000000  00000000   *$. . . . . . . . . . . . . . . *

:+1
G=00:1, T=02, S=02, PS=07

:d
00  00584449  52202020  20434F4D  0000000C   *.XDIR   COM.  .  .  .  *
10  27280000  00000000  00000000  00000000   *'(. . . . . . . . . . . . . . . *
20  00445520  20202020  20434F4D  00000020   *.DU     COM.  .  .  .  *
30  2526292A  00000000  00000000  00000000   *%&)*. . . . . . . . . . . . . . *
40  00434B2D  46495820  20434F4D  00000004   *.CK-FIX COM.  .  .  .  *
50  2B000000  00000000  00000000  00000000   *+. . . . . . . . . . . . . . . *
60  00505249  4E542020  20434F4D  00000004   *.PRINT  COM.  .  .  .  *
70  47000000  00000000  00000000  00000000   *G. . . . . . . . . . . . . . . *
```

```
:+1
G=00:2, T=02, S=03, PS=OD

:d
00  00434F4D 50415245 20434F4D 0000000A  *.COMPARE COM. . . . *
10  2C2D0000 00000000 00000000 00000000  *,-. . . . . *
20  0043504D 20202020 20434F4D 00000046  *.CPM     COM. . . F*
30  51525354 55565758 59000000 00000000  *QRSTUVWXY. . . . . *
40  00504950 20202020 20434F4D 00000037  *.PIP     COM. . . 7*
50  5A5B5C5D 5E5F6000 00000000 00000000  *Z[®]©_°. . . . . *
60  00535542 4D495420 20434F4D 0000000A  *.SUBMIT  COM. . . . *
70  61620000 00000000 00000000 00000000  *ab. . . . . *

:+1
G=00:3, T=02, S=04, PS=13

:d
00  00435255 4E322020 20434F4D 000000080 *.CRUN2   COM. . . . *
10  2E2F3031 3233B2B3 B5B6B7B8 B9BCBDBE *./0123. . . . . *
20  004D4241 53494320 20434F4D 00000080  *.MBASIC  COM. . . . *
30  34353637 38393A3B 3C3D3E3F 40414244  *456789:;< = >?@ABD*
40  004D4241 53494320 20434F4D 0100003B  *.MBASIC  COM. . . ;*
50  45464849 4A4B4C4D 00000000 00000000  *EFHIJKLM. . . . . *
60  00414C4C 4F432020 20434F4D 00000001  *.ALLOC   COM. . . . *
```

*NOTE: All numeric entries are in hexadecimal notation.

Figure 2.3. Dump of Sample CP/M Directory

2.3 THE SINGLE-DENSITY DISK

The single-density 8-inch disk uses sectors of 128 bytes. There are eight sectors in a group, so each group contains 1024 bytes. Each extent is 16 groups, so each extent contains 16,384 bytes. The first part of the directory is always group 0.

Physically, the single-density disk has 26 sectors (starting at sector 1) and 77 tracks (starting at track 0). Thus on a formatted disk there are:

$$\frac{128 \text{ bytes}}{\text{sector}} \times \frac{26 \text{ sectors}}{\text{track}} \times 77 \text{ tracks} = 256{,}256 \text{ bytes}$$

There are 26 × 77 or 2002 logical sector units available. The first two tracks, however, are used by the CP/M and are not actually available, so this leaves actually 26 × 75 or 1950 logical sectors available. Since there are eight logical sectors per group, this means 243 groups are available on a single-density CP/M floppy disk system.

Since there can be as many as 16 extents in a file and each extent can address 16,384 bytes, the maximum file size is 262,144—quite within the range of the single-density disk.

2.4 THE DIRECTORY ENTRY

The directory entry for each extent (Figure 2.4) consists of 32 bytes. This 32-byte entry is moved to the computer memory while the extent is being read or written. The entry in memory is called the FCB or *File Control Block*. The memory FCB is not identical to the directory entry, but is very close as we shall see later.

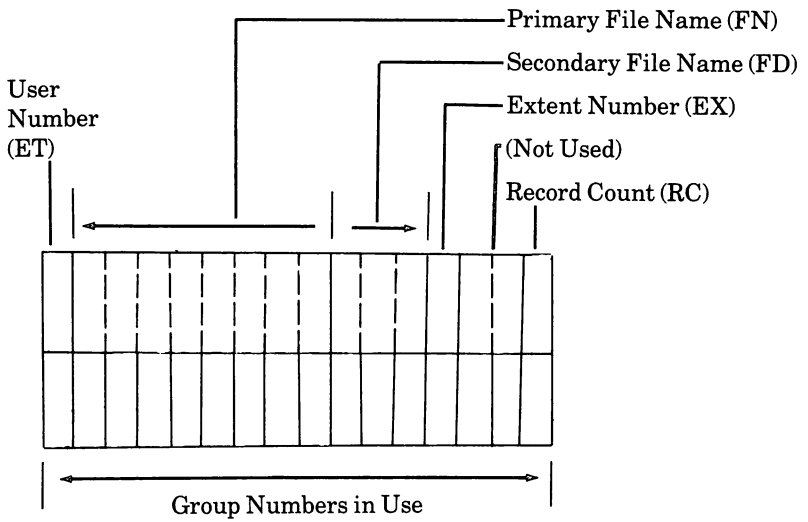


Figure 2.4. The Directory Entry

The first byte is the user number. In CP/M 1.4 this is 0; in later CP/M 2.2 and MP/M this is the user number for the file. Following this is the two-part file name and the extent number. The next two bytes are used by CP/M. The next is the *record count* or RC (the number of logical sectors used at this time) in the last group. When this reaches 80 (128), another group is allocated. The last 32 bytes are the group allocations for this disk. The complete disk group allocation for this disk is shown in Figure 2.6.

2.5 ALLOCATION STRATEGY

When a CP/M disk is logged into the system (Control/C), CP/M reads the disk directory and causes the BDOS to create a BIT map in memory that shows which group is currently being used. (Figure 2.5 is a bit allocation map for the disk mapped previously.) When you change disks, another Control/C must be given to create a new bit map and avoid an R/O error message. As the disk is written or erased, the bit map is constantly updated.

Group Allocation Map

```
   0123456789ABCDEF 0123456789ABCDEF
0  1111111111111111 1111111111111111  1
2  1111111111111111 1111111111111111  3
4  1111111111111111 1111111111111111  5
6  1111111111111111 1111111111111111  7
8  1111111111111111 1111111111111111  9
A  1111111111111111 1111111111111111  B
C  1111111111111111 1111111110000000  D
E  0000000000000000 000               F
```

Note: group number, use the row/column combination. For example, the next free group is DA, and the last free group is F2 (see Figure 2.6).

Figure 2.5. CP/M Bit Map for Disk in Figure 2.3

When a file is opened, the directory entry is used to create the FCB (file control block) in the computer memory. As new groups are added, this "memory image" FCB is updated along with the bit map. When the file is closed, the FCB is used to update the directory entry. In this way the directory of the disk almost always accurately reflects the file allocations even with system crashes. This is one of the most useful aspects of CP/M.

In other and more complex systems such as UNIX, Pascal, and even the older microsystems ICOM FDOS and MITS DOS, the practice of linking directory entries meant that linkages were usually lost on system crashes, making a file directory crash unrecoverable. With CP/M and a few simple utilities you can recreate directories and recover lost or erased files. There are also no packing (i.e., data compatification) operations necessary on the disk. The disk is always

packed! Packing programs, as they exist for CP/M, simply move the logical sectors of a file to sequential positions on a disk. This speeds up file reading, but gives the user no extra space. All free space is always immediately available to the user.

2.6 BOOTSTRAPPING CP/M

CP/M *bootstrapping* (or start-up) is a two-step process. A ROM chip in the computer loads a small program called BOOT from Track 0 of the disk and switches control to that program. The disk bootstrap, in turn, loads the rest of CP/M. The disk bootstrap must be altered depending upon the memory size. A *cold start* refers to the process of reloading the entire CP/M. Sometimes, however, it is only necessary to reload the CCP and BDOS. This is known as a *warm start*, and MUST be done whenever a disk is changed and you expect to write to the new disk (see Figure 2.7).

02-11	TEXX	.COM 0	:12-21	TEXX	.COM 1
22-23	TEXX	.COM 2	:24-24	D	.COM 0
25-26	DU	.COM 0	:27-28	XDIR	.COM 0
29-2A	DU	.COM 0	:2B-2B	CK-FIX	.COM 0
2C-2D	COMPARE	.COM 0	:2E-33	CRUN2	.COM 0
34-42	MBASIC	.COM 0	:43-43	ALLOC	.COM 0
44-44	MBASIC	.COM 0	:45-46	MBASIC	.COM 1
47-47	PRINT	.COM 0	:48-4D	MBASIC	.COM 1
4E-50	XREF	.COM 0	:51-59	CPM	.COM 0
5A-60	PIP	.COM 0	:61-62	SUBMIT	.COM 0
63-66	XREF	.COM 0	:67-68	MONTH	.ASC 0
69-6A	CARL	.ASC 0	:6B-6D	TASK	.BAS 0
6E-6F	RESOURCE	.COM 0	:70-71	TASK	.INT 0
72-73	RESOURCE	.COM 0	:74-74	FIND	.COM 0
75-75	LIST	.COM 0	:76-76	LINES	.COM 0
77-79	DUMP	.COM 0	:7A-89	OBASIC	.COM 0
8A-8B	OBASIC	.COM 1	:8C-91	ED	.COM 0
92-99	ASM	.COM 0	:9A-9E	DDT	.COM 0
9F-A0	LOAD	.COM 0	:A1-A2	MARGIN	.PRN 0
A3-A3	XMARG	.HEX 0	:A4-A4	MARGIN	.COM 0
A5-A7	STAT	.COM 0	:A8-A8	SYSGEN	.COM 0
A9-B0	COPY	.COM 0	:B1-B1	FORMAT	.COM 0
B2-B3	CRUN2	.COM 0	:B4-B4	D	.COM 0
B5-B9	CRUN2	.COM 0	:BA-BB	DISKTEST	.COM 0
BC-BE	CRUN2	.COM 0	:BF-BF	CRUN2	.COM 1

C0-CE	CBAS2	.COM 0	:CF-CF	MARGIN	.ASM 0
D0-D0	CBAS2	.COM 0	:D1-D4	CBAS2	.COM 1
D5-D6	RESOURCE	.COM 0	:D7-D8	FCOPY	.COM 0
D9-D9	SURVEY	.COM 0	:DA-FF	+ +FREE+ +	

Figure 2.6. Physical Map of Disk by Group Number

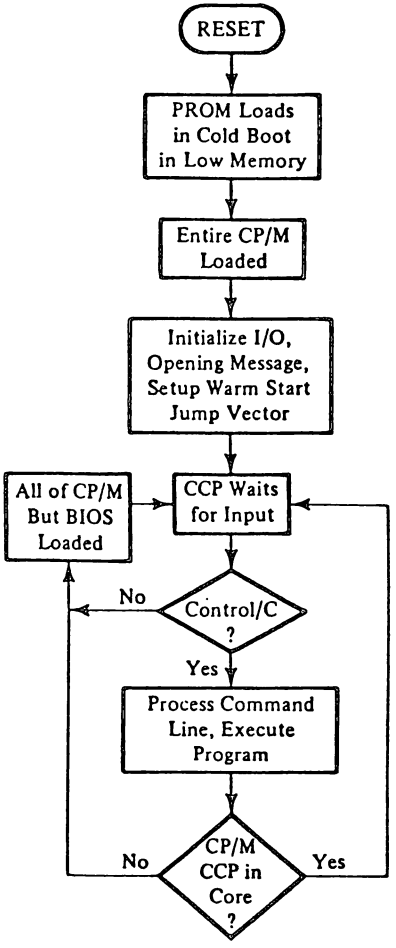


Figure 2.7. CP/M Bootstrap

Chapter 3

Interfacing to the IOBYTE

In wandering through your CP/M manuals, you've probably encountered the IOBYTE more than once. Very little is said about what it is, yet it seems to be important. Using the IOBYTE, you can switch output from the console to the printer or from one printer to another. Sounds exciting! Yet, when you tried it with STAT it didn't work. What went wrong?

3.1 WHAT'S THE IOBYTE?

A computer system normally has a large number of active peripherals (Figure 3.1). These peripherals are driven by a number of parallel and serial interfaces; each with a particular *port* or *I/O channel* and a separate, distrinct, *software driver*—that is, a program to send information to or receive input from that peripheral. The driver is called a physical driver, and in a CP/M system all of these physical software drivers are contained within the BIOS module.

In writing a computer program, however, you are scarcely concerned about interfacing to a particular peripheral. You're more concerned about high-level functions. For example, in debugging a program you might want to temporarily dump all the output that will eventually go to the printer onto the console. Once the program is working, you'll want to switch this output to the printer without changing the program.

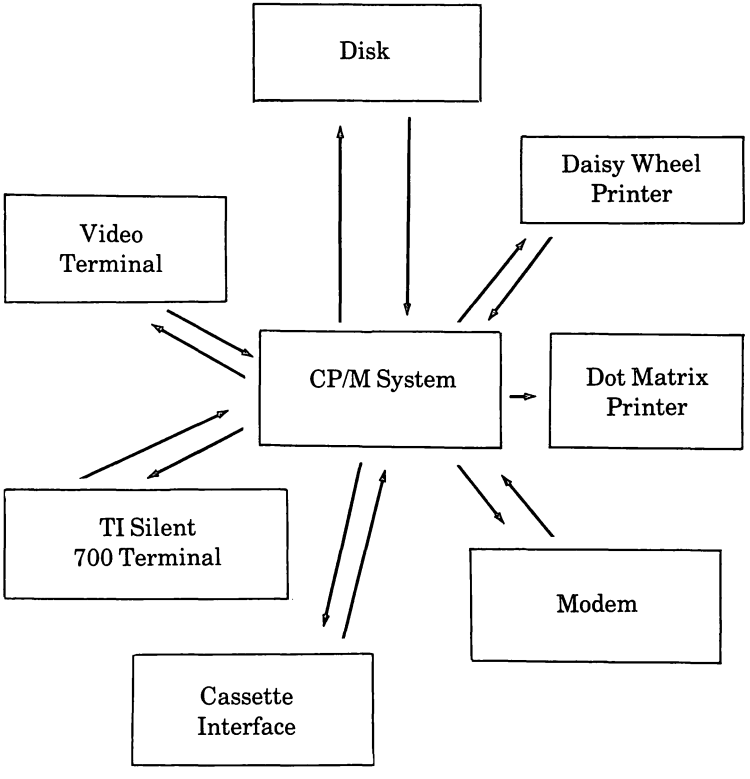


Figure 3.1. CP/M System With Peripherals

The software that performs each I/O function is called a *logical driver*. Thus, the main program interfaces to a functional or logical driver, which in turn can be connected to a wide variety of physical drivers (see Figure 3.2). Most of us don't have this many peripherals; we're showing them here to illustrate the difference between logical and physical drivers. Notice that the console logical driver is capable of two-way communication with peripherals. The other logical drivers provide only one-way communication.

The CP/M operating system provides four logical drivers (one for the console and three for other functions). Two of these are shown in Figure 3.2. These four can each be defined with three alphabetic characters and a colon:

CON: This function provides communication between the user and the system, and has provision for both input and output.

RDR: This function provides block data input from paper tape readers, modems, cassette players, card readers, or other CP/M systems.

PUN: This function provides block data output to a paper tape punch, cassette recorder, modem, card punch, or other CP/M systems.

LST: This function provides printing to a line printer.

The switches in Figure 3.2 allow a logical driver to be connected to any of several physical drivers. A physical driver can be connected to more than one logical driver at a time.

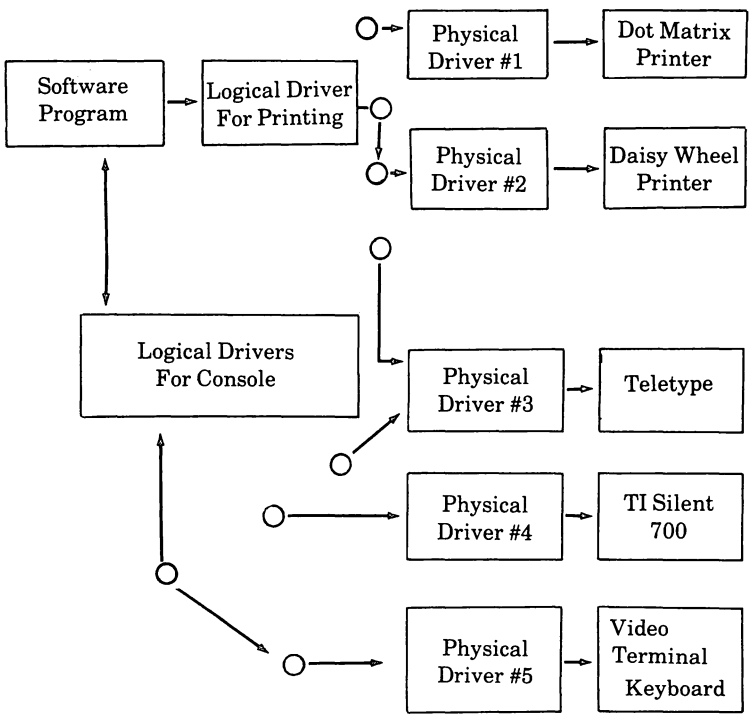


Figure 3.2. Logical Physical Interface

Alas, these wonderful switches do not exist in the distributed version of CP/M. CP/M provides for these switches and the concept, but if you want these in your system they must be implemented with software in the BIOS module. Some system manufacturers provide the user with a BIOS that already implements these switches. Other system manufacturers do not, and the logical drivers are "hard-wired" to specific physical drivers. In the latter case, if you wish to implement the switching function you must do it yourself.

3.2 THE FUNCTIONS OF THE IOBYTE

CP/M contains the software for its four logical drivers in the BDOS module. These drivers are interfaced to the physical drivers in the BIOS through a series of jump vectors at the beginning of the BIOS module. As an example, the CON: logical driver filters out any invalid control chracters as well as connecting to the LST: logical driver on a control/P toggle from the CON: device. The user can add additional software to the logical driver in the BIOS module as well as adding whatever physical drivers and switching is desired (see Figure 3.3). The CON: function has three jump vectors at the beginning of BIOS; the other logical drivers have one each

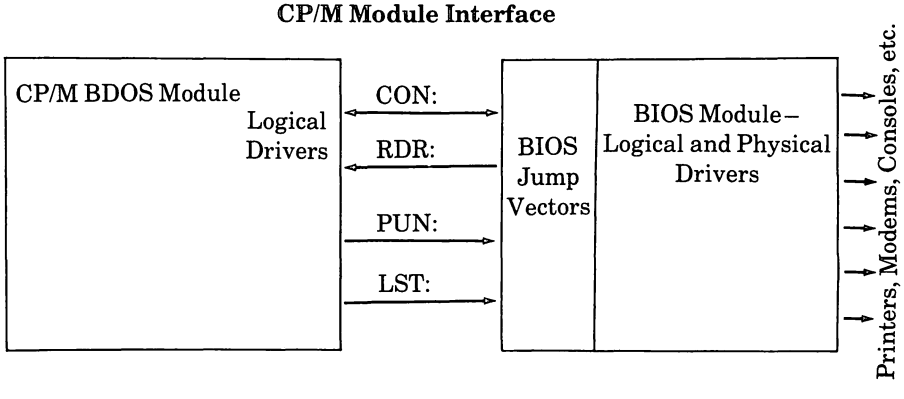


Figure 3.3.

(see Table 3.1). In a typical CP/M BIOS without IOBYTE, the jump vectors point directly to the physical device drivers.

Table 3.1. Mapping of Logical Drivers to BIOS Jumps

Logical Driver	BIOS Entry Point	BIOS Function
CON:	CONST	Checks console input status
	CONIN	Reads console input
	CONOT	Writes to console
LST:	LIST	Writes to list device
PUN:	PUNCH	Write to block-output device
RDR:	READER	Read from block-input device

CP/M allows you to use location 0003H as a software switch to determine which logical driver is connected to which physical driver. This location is called the IOBYTE. Two bits are used for each logical driver, so each logical driver can address any one of four physical drivers at any given time (see Figure 3.4). We now have four switches with each switch containing four positions. If you wish to use this function, software must be incorporated in the BIOS that looks at location 0003H and uses this information to "connect" each logical driver coming in through the jump vectors at the start of the BIOS to the correct physical driver. The distributed version of CP/M provided by Digital Research has only two provisions concerning the IOBYTE:

1. STAT already has within it the capability of altering the IOBYTE bit pattern in 0003H to conform with CP/M's physical device definitions. In other words, STAT can work the switch if you have the proper BIOS software to sense the switch.

2. PIP allows you to use physical device names for output or input. If you have the IOBYTE function implemented in your BIOS, PIP will channel its input and output to the correct physical device determined by the switch if you use a logical device name.

As stated earlier, either you or system manufacturer must alter the BIOS module to examine the IOBYTE and use the information stored in it correctly. This is not included in CP/M itself.

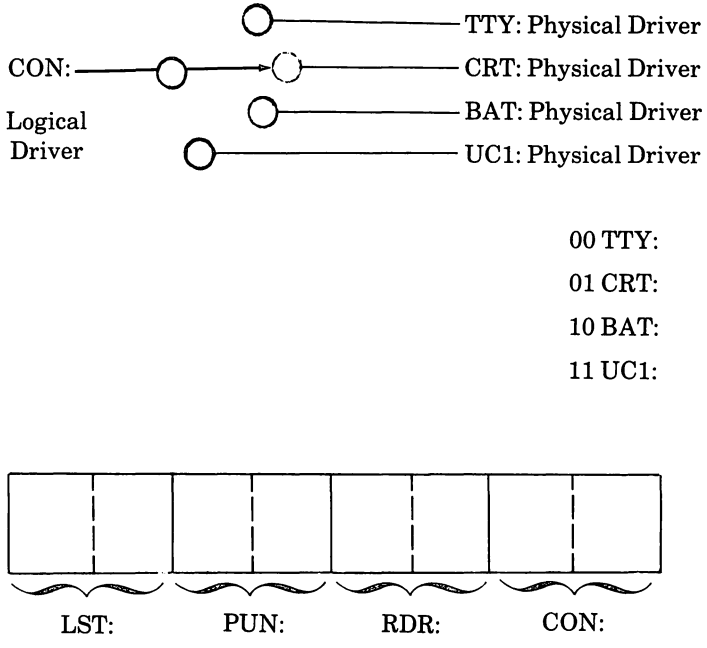


Figure 3.4. IOBYTE Switch for CON:

3.3 IMPLEMENTING THE IOBYTE

If the IOBYTE is not implemented in your system, you will find a straight-forward and simple implementation in a program by Jeff Kravitz called VBIOS31.ASM on the first volume of the CP/M User Group Library. In this program the logical drivers are separated from the physical drivers, and the IOCAL routine in the BIOS is used to isolate the bits of interest in the IOBYTE and determine which physical device to call.

IOCAL functions such as the I/O Dispatcher (see Figure 3.5). CONIN, CONST, and CONOUT are all part of the CON: logical driver that processes console input and output. All three of these must be switched together to the desired physical device. LIST, PUNCH, and READER are the other three logical drivers, each calling IOCAL. Each logical driver has very little logical function other than scanning the IOBYTE.

Sample of One Logical Switch

```
;                   CONSOLE   INPUT
;
CONIN:  LDA     IOBYT       ;GET IOBYT
        CALL    IOCAL       ;EXIT TO SELECTED ROUTINE
        DW      TTYIN       ;TTY INPUT ROUTINE
        DW      CRTIN       ;CRT INPUT ROUTINE
        DW      RDRIN       ;READER INPUT ROUTINE
        DW      NULIN       ;NULL INPUT ROUTINE
```

IOBYTE Dispatcher

```
;************************************************;
;
;                       I/O Dispatcher
;
;************************************************;

IOCAL:  RLC             ;SHIFT BITS
        ANI     06H     ;MASK BITS
        XTHL            ;SAVE HL, GET TABLE ADDRESS
        PUSH    D       ;SAVE DE
        MOV     E,A     ;GET SELECTION VALUE IN A
        MVI     D,0
        DAD     D       ;CALCULATE TABLE ENTRY
                        ADDRESS
        MOV     A,M     ;GET TABLE ENTRY
        INX     H
        MOV     H,M
        MOV     L,A     ;INTO HL
        POP     D       ;RESTORE DE
        XTHL            ;PUT EXIT ADDR ON STACK,
                        RESTORE HL
        RET             ;EXIT TO ROUTINE
```

Figure 3.5. BIOS Software to Process IOBYTE (From User's Group Volume 1)

Each logical driver loads the IOBYTE and rotates the bits of interest to the far right position. Note that for CONIN:, no rotation is needed. The logical driver then calls IOCAL, which examines the two rightmost bits and uses these to jump to the physical driver of interest.

On entry to IOCAL, the return address is pushed to the stack and the IOBYTE bits of interest are masked and dou-

bled. We double the IOBYTE code because each jump address in the table takes two bytes. XTHL at the beginning of IOCAL gets our return address (the first table entry) from the stack and stores this in registers H and L, storing what used to be H and L on the stack. The D register is saved on the stack and the IOBYTE, masked and doubled, is moved to D and added to the H and L registers with te DAD D instruction. The H and L registers are swapped with the stack pointer. Now when RET is executed, a jump is made to the desired physical driver.

Each logical driver functions similarly, rotating the desired bits of the IOBYTE to the far right and then calling IOCAL.

One other point should also be mentioned. In the part of BIOS that initializes the I/O ports for each physical driver, the IOBYTE should also be initialized (see Figure 3.6). Initialize it to whatever desired logical-to-physical mapping you desire.

```
SCPM:   XRA   A          ;INITIALIZE IOBYT
        STA   IOBYT
        MVI   A,OAAH     ;DUMMY MODE BYTE
        OUT   TTYS       ;CLEAR TTY
        MVI   A,040H     ;RESET COMMAND
        OUT   TTYS
        MVI   A,TTYMD
        OUT   TTYS
        MVI   A,TTYCM
        OUT   TTYS       ;SET UP TTY USART
        LXI   SP,80H     ;SET STACK
        CALL  DSINT      ;INITIALIZE DISPLAY
        LXI   D,CPMSG    ;POINT TO STARTUP MESSAGE
        MVI   C,09       ;WRITE BUFFER
        CALL  BDOS       ;CALL BDOS
        JMP   GOCPM      ;CONTINUE INITIALIZATION
```

Figure 3.6. Software to Initialize IOBYTE Processing

3.4 SWITCHING THE IOBYTE – USING STAT

If the IOBYTE is implemented, you will need to learn how to use STAT to alter location 003H. The four logical drivers have already been defined. CP/M suggests the following physical drivers:

TTY: slow speed teleptype device
CRT: high speed video console
BAT: batch processing (PRT: in, LPT: out)
UC1: user defined console
PTR: paper tape reader
UR1: user defined reader #1
UR2: user defined reader #2
PTP: paper tape punch
UP1: user defined punch #1
UP2: user defined punch #2
LPT: line printer
UL1: user defined list device #1

You can use these definitions in any desired manner, but these abbreviations are the only ones recognized and printed out by STAT and accepted by PIP. In my system, for example, I might hang a modem and a physical driver for the modem in my BIOS for PTR: and PTP:. In this case all references to these physical drivers would refer to the modem.

Now load a disk with the CP/M to your system and enter the following:

```
A>STAT VAL:
```

CP/M will give you the following output on the console (if you have a 2.X CP/M system, additional information about the system will also be displayed):

```
CON:  =  TTY:  CRT:  BAT:  UC1
RDR:  =  TTY:  PTR:  UR1:  UR2:
PUN:  =  TTY:  PTP:  UP1:  UP2:
LST:  =  TTY:  CRT:  LPT:  UL1:
```

This shows the four suggested physical devices to use with each logical device. Now enter:

```
A>STAT DEV:
```

This will show the correct status of the IOBYTE pointers:

```
.CON: is TTY:
 RDR: is TTY:
 PUN: is TTY:
 LST: is TTY:
```

Using DDT, we can actually look at location 0003H and see its value:

```
A>DDT
DDT VERS 1.4
-s3
0003 00 .
- (control/C)
```

Table 3.2. IOBYTE Mapping Definitions

Logical Device or Function	Physical Device			
	00	01	10	11
Console (CON:)	TTY:	CRT:	BAT:	UC1:
Reader (PTR:)	TTY:	RDR:	UR1:	UR2:
Punch (PTP:)	TTY:	PUN:	UP1:	UP2:
List (LST:)	TTY:	CRT:	LPT:	UL1:

The TTY: device is performing the CONSOLE function.
The PTR: device is performing the READER function.
The UP1: device is performing the PUNCH function.
The TTY: device is performing the LIST function.

Notice that the current IOBYTE value is 00. The full table of values is shown in Table 3.2. For the value of 00H, all logical values are set at TTY:. The line printer logical driver is set to the TTY: or console device. If the BIOS had the IOBYTE implemented, any output to the printer would go to the console device. If the BIOS does not have the IOBYTE implemented, output will go to the printer since most BIOS modules connect the physical printer driver directly to the LST: jump vector. If the BIOS does have the IOBYTE implemented, anything sent to the printer will go to the console instead, since the BIOS will connect the printer jump in the BIOS to the console physical drivers.

Now use the STAT to change the logical device assignment for LST: to the printer:

```
A>STAT LST:=LPT:
```

Now check the assignments:

```
A>STAT DEV:
CON: is TTY:
RDR: is TTY:
PUN: is TTY:
LST: is LPT:
A>DDT
DDT VERS 1.4
-s3
0003 80 .
- (control/C)
```

The value of the IOBYTE is now changed to reflect the new mapping. If the switching function is implemented in the BIOS, the printer output will now go to the printer.

Chapter 4

Using PIP and SUBMIT

PIP is a CP/M utility for moving files from one location to another. The initials stand for Peripheral Interchange Program. It allows the transfer of files between any two devices. This could be disk to disk, disk to printer, disk to a remote computer, disk to console, etc. Simple modifications and file verification can be made during the transfer. For the non-technical user, PIP is probably one of the most frequently used utilities. Using PIP you can:

a. Merge several files in a single file.

b. Create backups of programs and data files.

c. View a file on the console up to a specific point without using the editor.

d. Transfer small ASCII files to or from a remote computer.

e. Print files with formatted pages and tabs.

4.1 TWO MODES

PIP can be used in either of two modes. For a single file transfer (or the transfer of a single group of files) the command line can contain the complete transfer definition. For example:

A>PIP B:=A:*.BAS[vo]

will transfer all BAS files on the A drive to the B drive and verify each file after transferring. If more than one file is

transferred, PIP will display the name of each file during the transfer. The "v" option requests that the copy be verified after completion. The "O" option should be used on all non-ASCII files; it instructs PIP to ignore any CONTROL/Z's that may be in the file. CONTROL/Z is normally used as the end-of-file character for CP/M files. With non-ASCII files, however, a CONTROL/Z may be a part of the data and should not be used to signal the end-of-file. This causes all of the space allocated in the directory for the file to be copied to the new file even though it is not all used.

In the second mode, PIP can be called as a program using:

```
A>PIP
```

Once PIP has loaded, an asterisk will be displayed. PIP commands can then be entered. For example:

```
*B:=A:.BAS[vo]
```

After the transfer is complete, the asterisk will be dislayed again. You can enter another PIP command, or a Control/C to exit PIP.

To batch jobs, using submit, it is necessary to use the first mode and load PIP for each transfer with CP/M 1.4 or MP/M as:

```
A>PIP    D:=B:*.*[vo]
A>PIP    C:=A:*.*[vo]
```

With CP/M 2.0, XSUB can be used to avoid reloading PIP each time. In this case, all the files of B are transferred to D and all the files of A are transferred to C.

PIP does not do a warm boot on starting. If the disks have been changed without doing a warm start; you will get a R/O error message if you try to write to the changed disk and you will have to restart. To prevent this:

1. Load the system disk with PIP to the A drive; the destination disk is B.
2. Boot the system and enter "PIP" and a carriage return.
3. Remove the system disk. Load the source data disk to "A".
4. At the console enter the desired command and a carriage return.

Notice that although the disk in A has been changed, A will only be used for reading and will, therefore, not generate the R/O message. The destination disk has not been changed.

4.2 PIP WITH DEVICE NAMES

PIP can also be used with either physical or logical device names. For example, to send a file to a printer you could use:

```
*LST:=TEST.TXT
```

In addition, as we shall see in the next chapter, PIP can be used to transfer files between computers using the block mode transfer as in:

```
*PUN:=TEST.TXT,EOF:[B]
```

Table 4.1 shows the logical device names that can be used in any PIP transfer; Table 4.2 shows the physical device names that are also permitted. Table 4.3 shows special device names that can also be used in any transfer. The EOF:, for example, transfers an end-of-file character after the file has been transmitted.

You can only send text (ASCII) or hex files to printers and consoles.

Table 4.1. PIP Logical Devices

CON:	console
RDR:	reader (or block read input device)
PUN:	punch (or block read input device)
LST:	list device

Table 4.2. Physical PIP Devices

TTY:	teletype
CRT:	video terminal
BAT:	console input, list output
UC1:	user-defined console
PTR:	paper tape reader
UR1:	user-defined reader #1
UR2:	user-defined reader #2
PTP:	paper tape punch
UP1:	user-defined punch #1

UP2:	user-defined punch #2
LPT:	line printer
UL1:	user-defined list #1

Table 4.3. Special PIP Parameters

NUL:	send 40 nulls to device
EOF:	send EOF (control/Z) to device
INP:	special input device
OUT:	special output device
PRN:	same as list except tabs are converted to eight spaces, lines are numbered, and page ejects are added

Table 4.4. PIP Parameters

B	Specifies "block mode" transfer.
Dn	Deletes all characters after the "n"th column.
E	Echoes the copying to the console as it is being performed.
F	Removes form feed characters during transfer.
Gn	Directs PIP to copy files from other user areas.
H	Checks for proper Intel Hex File format.
I	Ignores any ":00" records in Intel Hex File transfers.
L	Converts upper-case letters to lower-case.
N	Adds a line number to each line transferred.
O	Object file transfer (ignores end-of-file markers).
Pn	Issues page feed after every "n"th line.
Qs ^Z	Specifies end (Quit) of copying after the string "s" is encountered.
R	Directs PIP to copy from a system file.
Ss ^Z	Specifies start of copying after the string "s" is encountered.
Tn	Sets tab stops to every "n"th column.
U	Converts lower-case letters to upper-case.
V	Verifies copy by comparison after copy finished.
W	Directs PIP to copy onto a R/O file.
Z	Zeros the parity bit on ASCII characters.

4.3 SPECIAL TRANSFERS AND PARAMETERS

PIP can also be used to concatenate files. For example, the expression:

```
>PIP TEST.TXT = TEMP1.TXT, TEMP2.TXT,
 TEMP3.TXT
```

will concatenate TEMP1.TXT, TEMP2.TXT, and TEMP3.TXT to a single TEST.TXT file.

The assembler produces a hex file that is converted to a COM file with LOAD. Hex files can be concatenated or copied with the H or I parameter. With this option, PIP will verify and check the HEX format for errors.

Optional parameters can be used in the copy operation to control the transfer. These are listed in Table 4.4. For example, the command:

```
LST:=STARTREK.BAS[NT8P60]
```

will print the file STARTREK.BAS with line numbers, tabs at every eighth column, and page ejects every sixty lines. If PRN: were used here instead of LST:, none of these parameters would have been needed. PRN: assumes these by default. The resulting command would be:

```
PRN:=STARTREK.BAS
```

Using the "Q" and "S" options, portions of files can be printed or copied as:

```
A>PIP PRN:=TEST.BAS[QFLAG↑Z]
```

would print everything up to "FLAG". If a lowercase "flag" is entered, it is translated to uppercase. If PIP were used in a program mode, the search string would not be capitalized and the search would be for "flag."

4.4 ENHANCEMENTS IN CP/M 2.2 AND MP/M

MP/M and CP/M 2.2 permit the user to separate the disk(s) into several logical areas for different users or projects. Normally, one user cannot access files of other users except those in user area 0. All users have access to user area 0. If you have a single-user system, it is not necessary or even advisable to use other user areas than area 0 unless you are trying to read a disk created by another user on an MP/M system. If you are using an MP/M system, you will need to use these additional enhancements.

A second feature of CP/M 2.2 systems and MP/M systems is that you can designate a file as R/O (Read Only). Once designated as R/O, the file cannot be written over (changed or

deleted). To alter such a file, it is first necessary to use PIP to change the file to R/W (Read/Write). It is also possible in CP/M 2.2 and MP/M systems to set a file as a "system" file. This gives additional protection.

The additional parameters for PIP in CP/M 2.2 and MP/M are the following:

Gn	Get file from user area "n"
W	Overwrite an R/O file
R	Copy a file with a "system" attribute

For example, the command:

```
A>PIP TEST.TXT[G3]
```

would copy a file from user area 3 on a disk to user area 0. TEST.TXT would then exist on the disk as *two* files–one in user area 3, the other in user area 0. A copy of PIP must be in each receiving user area. In this example, a copy of PIP must be in user area 0. This creates no problem in CP/M 2.X systems, but in MP/M systems you will need some expertise to move PIP to different user areas before files can be transferred to these user areas.

4.5 USING SUBMIT

Often it would be useful to store a group of commands and then to execute this group with a single command from the console. This is termed *batch processing* and can be done in CP/M with SUBMIT. Typical applications include:

Chaining Procedures–Some procedures are lengthy and it is easy to make a mistake somewhere in the serial process. An example would be compiling a RATFOR source program to a FORTRAN source program, compiling the FORTRAN program, and then linking the relocatable module to an executable COM file which is finally called for execution. This procedure could be stored and executed as often as desired in a single submit file.

Complex Command Storage–Some programs require a complex command string on startup. To start a KBASIC program, for example, a typical command might be:

```
A>KBASIC MENU /F:15/x:4
```

To simplify the process to an inexperienced user, the entire command could be stored as a submit file with the start up as:

```
A>SUBMIT MENU
```

Program Installation—Some programs, such as SELECT, are sold with a SUBMIT file that creates a customized copy of the master program for the user's system. This minimizes the chance of error during the installation process.

Multiple File Backup—Using SUBMIT, a master program can be written that will back up all user files to a second disk. The program can then be run by an inexperienced operator or scheduler with a minimum chance of error and the assurance that everything will get backed up.

To start SUBMIT, use the editor to build a file with the secondary name of ".SUB" such as "BACKUP.SUB". This file should have all the commands to be processed; for example:

```
PIP D:=B:*.*[vo]
PIP C:=A:*.*[vo]
```

All commands should be direct. The text:

```
PIP
B:=c:TEST.BAS[vo]
```

would not be valid in a SUBMIT file:

Once this file is built, call this file with submit *without* the secondary name:

```
A>SUBMIT BACKUP
```

SUBMIT copies BACKUP.SUB to the disk as a new file called $$$.SUB. The system then does a warm boot, but the CCP uses this temporary file as input instead of the keyboard. Commands are read serially from this file and executed. Upon completion, control is returned to the keyboard. Several additional factors should be noted:

1. In starting SUBMIT, you MUST be logged on to the A disk. The $$$.SUB file is ALWAYS created on the logged on disk and the system then reboots to this file. If the

$$$.SUB is created on a different disk, the system will not find it on rebooting, but it will find it the next time you put the disk with the $$$.SUB file on the A drive. In other words, on booting, CP/M always checks first for a $$$.SUB file; if it finds this file, it uses it instead of the keyboard for input. You could, however, use a submit file on B and a SUB file on B as:

```
A>B:SUBMIT B:COMPILE
```

2. The SUB file, as mentioned, must contain only direct commands. The series:

```
PIP
B:=A:TEST.ASC
```

will not work.

3. If the SUBMIT program is terminated prematurely with Control/C, the $$$.SUB file will remain on the disk. The next time the disk is rebooted, the $$$.SUB file will gain control. The batch execution can be stopped and the $$$.SUB file erased with the RUBOUT key.

4. Any running SUBMIT program can be terminated with RUBOUT.

5. Control characters can be used in command strings but should be stored using the "↑" arrow instead of the control key. For example, "Control/P" would be entered as "↑P".

With CP/M 2.2 or later, XSUB can be used to extend batch processes to include commands to programs. Simply include the XSUB as the first statement in the SUB file, and then add program inputs as well as commands. For example:

```
XSUB
PIP
B:=A:TEST.ASC
```

is valid with CP/M 2.X.

4.6 SPECIAL TRICKS USING SUBMIT

Control characters cannot be used in a SUBMIT file; but you can store control characters by performing a substitution of the form:

"⟨CHAR⟩"

in the command line. Using this trick, you can switch the printer on and off from a batch file.

You can put comments in a SUBMIT file by starting the comment line with a colon. A semicolon can also be used; but Digital Research documentation supports only the use of the colon.

SUBMIT jobs can also be chained with one SUBMIT job calling another. The call to the next SUBMIT, however, must be at the END of a batch job. The ASSEM.SUB file:

```
SUBMIT STARTUP
ASM TEST
```

would not work correctly, as the first command line would erase the temporary file created by executing ASSEM.SUB.

The batch process can be speeded up by putting an empty space at the beginning of a disk that you plan to use for batch processing. When a SUBMIT file is executed, the temporary file is built with the first command *last*. As each command in the temporary file is executed, it is erased from the temporary file from the bottom up. To get each command from the temporary file, the CP/M goes to the end of the temporary file. The SUBMIT process can be speeded up by putting the temporary file near the beginning of the disk next to the directory.

When creating a disk you intend to use for batch jobs create an empty file with the SAVE command that is large enough to hold the temporary file. Normally, not much space is needed, for example:

```
A>SAVE 1 TEMP.SUB
```

Do this *first* before storing any other files on the disk. Just before running your SUBMIT, erase this file. This will leave an empty space near the beginning of the disk.

Chapter 5

Talking to Other Devices

Communication between your microcomputer and peripheral devices such as keyboards, printers, disks, and other computers can be any of three types:

1. program controlled
2. interrupt controlled
3. Direct Memory Access (DMA) controlled

The type of communication control for each peripheral in your computer system is determined by how the hardware is designed and how your particular BIOS module is written. The manufacturer of a given CP/M system designs the BIOS module to support the associated hardware. The CP/M BIOS for version 1.4 can be written to support program, interrupt, or DMA control. Later CP/M releases are moving toward requiring interrupt or DMA control. This means that you should know something about each type of control and how it relates to your hardware, the CP/M operating system, and your own system needs.

5.1 PROGRAM CONTROLLED I/O

Program Control is the most popular type of peripheral control used by hardware manufacturers. With this type of control, input is accepted from the keyboard only when the computer wants input. The program can only do one thing at a time. Therefore, if the printer is printing, you can't do anything else until the printer is finished. If a sector is being

written to a disk, again you must wait until the disk writing is complete for any other operation. Since the computer is quite fast, most users don't miss the lost time. The problem only becomes apparent if your computer is communicating with another computer or if more than one person is trying to use the system.

Because program-controlled communication is so popular in low-cost systems, various methods have been developed to make it faster and easier to use. Spoolers for printers will use "free" processor time to print while you are doing other jobs. Handshaking techniques can be used in communications between computers to minimize the problems of interfacing the simplified hardware. All solutions, however, involve some loss of efficiency and time in the use of your system. Maximum efficiency is only available with interrupt driven systems and disk DMA interfacing.

If you are using CP/M version 1.4, someone experienced in programming can tell from the BIOS listing whether your system I/O is program controlled or interrupt controlled, and whether the disk is DMA interfaced or not. For example, you may have a statement:

```
INTRP EQU TRUE
```

near the beginning of the BIOS. This doesn't necessarily mean, however, that your system is using interrupt control. The interrupt may be turned on but never used. The DMA, in the same way, may not be used even though there are variables in the BIOS referencing the DMA. Check with a programmer if you really want to know.

My own microcomputer uses program control. This gets to be quite a problem when using The Source or other electronic communication systems. Suppose, for example, the remote computer is sending data at a slow 300 baud rate. My computer stores it until there is enough data to write a sector (128 bytes) to the disk, then writes to the disk. However, during the time that the computer is writing to the disk, one or two more characters have arrived from the remote computer. Since the computer can't read and write at the same time, these characters are lost.

One way to resolve this is to instruct the remote computer to only send one page or data block at a time and then stop and wait for a signal. You then enter a control character that

transfers this block or page to the disk. After this transfer, enter a signal to start the remote computer again. Other programming tricks will also work here, but the problem is always the same. Transmission time and efficiency are lost.

5.2 INTERRUPT CONTROLLED I/O

In this method of communication control the computer does nothing until it is told. The program runs in an "idle loop" until it is interrupted by a peripheral such as the keyboard. When the computer senses the interrupt, it jumps to a special program to process the interrupt. After the processing is done and any command given is acted upon, the computer returns to an idle loop to wait for the next interrupt. If a second interrupt from another peripheral arrives while the computer is busy with the first, a priority scheme determines which is more important. Highest priority is generally given to the fastest device–often the disk.

Interrupt control is the option of the manufacturer with CP/M version 1.4, and will be an integral and necessary part of future CP/M versions. In the better CP/M systems, all slow-speed I/O (such as the video and printers) are interrupt driven.

5.3 DMA CONTROLLED I/O

DMA (Direct Memory Access) is a transfer method whereby information is transferred directly to or from the computer memory without going through the processor. The advantage is obvious–the processor can be doing other things during this transfer time. This transfer method is used with hard disk systems and the better floppy disk systems. In combination with interrupt controlled I/O for lower speed peripherals (such as keyboards, printers, and modems), it permits the hardware to operate at maximum speed and efficiency. DMA can be used for CP/M disk transfer and will be required in future CP/M versions.

In the DMA scheme, the DMA controller uses the time when the system bus is not used by the processor to transfer information directly to or from memory. The processor tells the DMA controller the starting memory address for the transfer and the number of bytes to transfer. The DMA con-

troller takes this information, senses when the bus is not being used, and transfers data to or from the disk during this time. The processor remains available to the programmer for the next job (Figure 5.1).

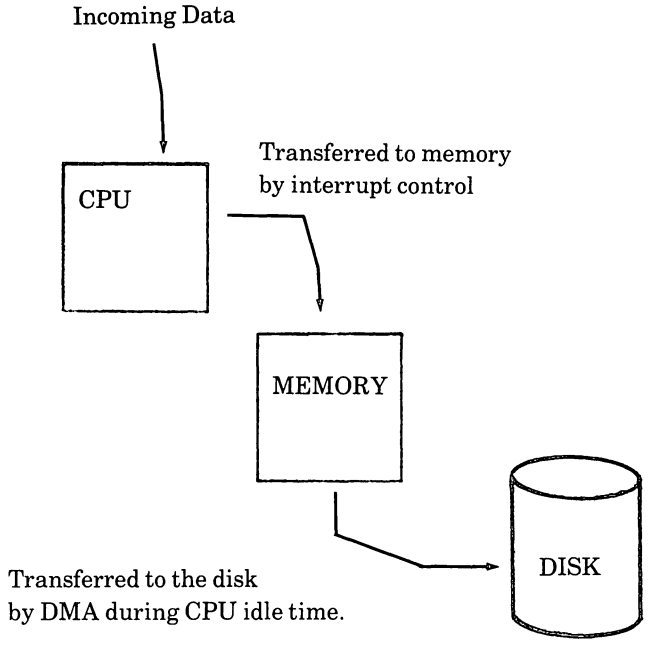


Figure 5.1. DMA Transfer

For example, suppose you are transferring data from a remote computer, through a serial port and modem, to a file on your system. In a DMA system, the modem transfer program in the transient program area (TPA) fills a buffer area in the computer memory with the incoming data. When the buffer is full, the data in this area is block-transferred to a special disk buffer on the disk controller card and the DMA begins transferring the data from this area to the disk. During this transfer the modem program can continue to receive data and store it in the internal buffer. It is important that the DMA controller has been designed not to slow the processor during the transfer. In this way the computer can keep

up with incoming data, even though periodically this data is transferred to the disk. There are several ways to do DMA, and some are better than others. In the more efficient DMA systems, the CPU loses no time during the disk transfer.

5.4 APPLICATION AREAS

The three primary application areas for communication between microcomputers are (see Table 5.1):

1. Moving programs and data from one computer (such as a small mini-disk system) to another (such as a computer with larger floppies or a hard disk). The computers are hard-wired together.
2. Electronic mail and communication systems (such as The Source and CompuServe) using telephone lines.
3. Distributed intelligence (Local Area Networks or LAN). An example of this might be a network of small computers inside a company linked together for word processing or inventory control. Coaxial cable or twisted pair cable is used.

Table 5.1. Communication Classifications

Application	Connection Type	Distance	Speed	Software
File transfer between computers	Direct or RS-232	1–500 meters	100–10K bits/second	PIP or special programs
Electronic mail or bulletin boards	Modem and telephone line	(unlimited)	100–1200 bits/second	Modem or bulletin board programs
Local Area Networds (LAN)	1000 meters	To 1000 meters	To 10 megabits/second	CP/NET

Each of these is a distinct application area and has its own unique requirements.

Sharing programs and data in ASCII format is often done over an RS-232 serial data line. In many cases, if the file is

small, it can be directly transferred with PIP. However, recall that only ASCII files can be transferred by PIP. To transfer a COM file, first convert the file to HEX format using MP/M's GENHEX or the User Group's UNLOAD Program (Volume 29). Since the hex file is an ASCII file, it can be transferred with PIP.

To use PIP, generate a version of CP/M for the receiving system with a serial input port such as the reader device. For the source computer, the serial output port should be the punch device, but the list output (if serial) is often used. The receiving computer is set up as:

A>PIP RDR: = test.asc

where: test.asc is the name of the received file on the disk.

The transmitting computer is set up as:

A>PIP PUN: = test.asc,EOF:

Note too that the PIP parameters H and I can be used with HEX files. The "H" parameter is used to check HEX files on transfer for proper format. The "I" parameter will enable the transfer to ignore ":Q0" records.

PIP uses the entire memory space of the destination computer as a buffer. If the file is very long it may completely fill the buffer. When this happens, the computer stops receiving data to write the buffer to disk. By the time the computer is ready to receive again, considerable data may be lost.

If the transmitting computer recognizes the PIP option "B," it can be used to prevent data overflow at the receiving computer. Option B forces an automatic X-on and X-off dialogue (control/Q and control/S) between the computers to prevent the receiving buffer from overflowing.

Good modem software will have built-in protection from buffer overflow using the X-on and X-off controls. When the receiving buffer is nearly full, the software automatically sends the source computer a control code. This stops the transmission and permits the receiving buffer to be emptied to disk. Then the destination computer resumes the transmission by sending another control character. Both The Source and CompuServe recognize this protocol. Bulletin boards often have their own protocol which varies from system to system.

The second application area, electronic mail, requires special software and a modem. The modem converts the RS-232 digital information to an analog signal for transmission via telephone lines. Modems can be purchased for as little as $120, but $200-$300 is the normal price for a 300-baud modem. This information transfer rate is about the rate you can read, or about twice the speed of a high-speed Selectric typewriter. Some software is available through the CP/M User Group (Volumes 25, 46, 47), and this is a good starting point. Any modem software purchased should support the X-on and X-off protocol to prevent loss of data during disk storage. Using DMA for the disk is also highly advised.

The third application area, distributed intelligence, requires multi-user software and very high-speed connecting lines. Data is moved on a continuing basis at rapid transfer rates between systems. Disks, printers, and other peripherals are shared by users. Both the hardware and software for this application are expensive at the present time, but costs are rapidly dropping.

Chapter 6

CP/M I/O Function Calls

CP/M was originally written in PL/1 and compiled to 8080 code (CP/M-86 is compiled to 8086 code. And CP/M will almost certainly be compiled eventually to other microcomputers.) Plenty of books are available on 8080 assembly language coding, but little information is available on interfacing your assembly-level source programs to CP/M. This will be the subject of this chapter.

6.1 I/O INTERFACING FOR CONSOLES AND PRINTERS

There are two ways to do I/O operations in assembly-level language routines (see Figure 6.1). One method is to jump directly to the desired routine call at the beginning of the BIOS. The other method is to use the BDOS calls (as explained in the *CP/M Interface Guide* published by Digital Research) to interface with the I/O routines in the BIOS. The two methods each have advantages and disadvantages. Let's take a closer look at each approach.

The I/O routines in the BIOS are called *primitives*. These routines perform the same *function* in any system, and as such are functions that are independent of the system hardware. CONOUT always outputs to the console device and SETTRK always seeks to the track in register A. The actual code for these routines will vary with the hardware, but the function is always the same.

These primitive routines can be used directly as they are in the BIOS. The normal method is to call them through the

BDOS. Using BDOS is the normal, preferred approach if at all possible. This will make your software "portable" so it can be run on a wide variety of CP/M systems as well as give you some file protection with disk I/O. In the next two sections we will talk about I/O using peripherals as consoles and printers and show you both methods of interfacing. Then we will expand these concepts to include the disk.

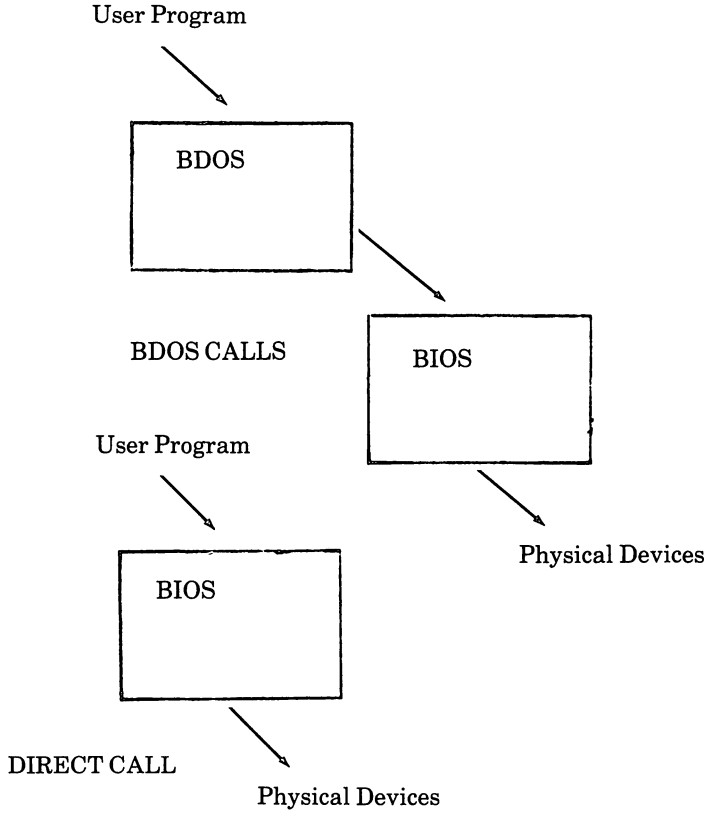


Figure 6.1. BDOS Call vs. Direct Call

6.2 BDOS CALLS

The BDOS could be considered a filter for calls to the I/O. Error characters (such as unwanted control characters) are filtered and functions like control/S and control/P are imple-

mented. Programs using BDOS calls behave like other CP/M programs and programming work is simplified.

The location of the BDOS is *always* stored in CP/M memory at locations 5, 6 and 7. This is true for any size CP/M system and versions 1.4 and 2.2. The user simply calls to this location. There is, however, a certain protocol that must be followed:

1. Store any stack pointer you are using and *all* registers before calling.
2. The number for the function desired (see Table 6.1) is always in register C.
3. Other registers are used to store values used in the transfer (see Table 6.2).
4. Restore the stack pointer and registers upon returning to your program.

More details of using BDOS calls can be found in Digital Research's *CP/M Interface Guide.* For example, to read a console character use the following code:

```
fdos   equ  5
rdcom  equ  1
readd: mvi c,rdcon
       call fdos
```

The character will be returned in the accumulator.

Table 6.1. CP/M I/O Functions

(Note: Function Code is placed in register C before call.)

Function Number	Function
1	Read one character from console
2	Write a character to the console
3	Read character from reader
4	Write character to punch
5	Write character to printer
*6	Direct console I/O
7	Get I/O status
8	Set I/O status

9	Output string to console
10	Input string from console
11	Interrogate console to see if character is ready
*12	Return version number

*available CP/M 2.2 only

Table 6.2. I/O Functions and Register Usage

Function Code (in Register C)	Function	Entry Parameters (Registers D&E)	Returned Value in Registers A or A and B
1	Read one character from console	None	ASCII characters
2	Write one character to console	ASCII character	None
3	Read character from reader	None	ASCII character
4	Write character to punch	ASCII character	None
5	Write character to list drive	ASCII character	None
6	Get I/O status	None	I/O status byte
7	Set I/O status	I/O status byte	None
8	Output string to console	Address of print string	None
9	Input string from console	Address of read buffer	(Read buffer loaded)
10	Interrogate console status	None	01 if ready

Figure 6.2 shows a short program for modem transmission. It is a modified version of the program listed in Appendix I. The modem calls are direct and within the program; they do not go through the BIOS. All of the console and printer calls, however, are BDOS calls.

```
FILE: GETVEC   LIB    PAGE 001

;*****
;  OBTAIN BIOS VECTORS
;*****
VECTRS:  JMP    GETVEC
         DS     42
WBOOTE   EQU    VECTRS+3
CSTS     EQU    VECTRS+6
CI       EQU    VECTRS+9
CO       EQU    VECTRS+12
LO       EQU    VECTRS+15
PO       EQU    VECTRS+18
RI       EQU    VECTRS+21
HOME     EQU    VECTRS+24
SELDSK   EQU    VECTRS+27
SETTRK   EQU    VECTRS+30
SETSEC   EQU    VECTRS+33
SETDMA   EQU    VECTRS+36
READ     EQU    VECTRS+39
WRITE    EQU    VECTRS+42
GETVEC:  LXI    D,WBOOTE
         LHLD   1
         MVI    B,42
GETVE1:  MOV    A,M
         STAX   D
         INX    H
         INX    D
         DCR    B
         JNZ    GETVE1
```

Figure 6.2. Capturing BIOS Vectors from a User Program for Direct BIOS Call (from User Group Disk #1)

To display or read strings, a buffer is used. To display a buffer, store the string in sequential addresses:

n1 n2 n3 n4–nK$

and set the D and E registers to point to the starting address. The "$" sign always terminates the output. For input provide a buffer:

i j n1 n2 n3 n4 n5–nK

where i = maximum number of characters to be input
j = actual number input (managed by CP/M)
ni-nk = input characters

Appendix I includes a good example of this. Table 6.1 contains a list of all BDOS console and printer calls.

6.3 DIRECT I/O

It is also possible to use the BDOS primitives directly. The beginning of the BIOS is a jump table to each of the primitives in a pre-defined order. This table is defined identically for all CP/M systems. These jumps, or vectors, can be copied to your program and used as I/O calls for your own program. A good routine for doing this is GETVEC.LIB in the CP/M User Group Volume 1 (see Figure 6.2 and Appendix F). This routine gets the location of the warm boot vector (the start of the BIOS) from addresses 1 and 2 in the CP/M memory. The remaining vectors start from this vector and can be copied to your own program and used.

This approach is often used if special control codes that would normally be filtered by the BDOS must be passed to a device. An example would be using your terminal and computer with a remote computer. The remote computer might need important control codes that CP/M BDOS calls would not pass. This approach is also used to obtain special access to tracks or sectors of the disk–for example, in a program to alter specific sections of a disk or copy disks sector by sector and track by track. There is, however, one serious disadvantage. Programs using direct I/O are occasionally not compatible between CP/M versions or between CP/M and MP/M.

One of the reasons for this is that while using the BDOS you see a logical system that remains compatible between any type of CP/M or MP/M system–at least it will be upward compatible. Using direct calls, however, this logical system is bypassed and you actually see a "hardware" system.

Examples of direct I/O programs include the DU and SAP programs in the CP/M User Group Library. Both are valuable programs, and DU can save a great deal of time by enabling you to restore files that were erased by mistake (more on this later). Both of these programs, however, use direct I/O calls and are not compatible with some CP/M systems.

6.4 BDOS VERSUS DIRECT CALLS

Table 6.3 is a comparison of BDOS call I/O and Direct I/O. These types should not be mixed in a program. BDOS calls offer portability and compatibility. Direct I/O offers freedom from CP/M constraints and definitions, as well as hardware access that bypasses logical conventions.

As a general rule, you should use direct BIOS calls if you are:

- making direct modifications of a disk directory (for example, restoring an erased file or patching a bad directory)
- accessing information by a direct call to a track and sector (for example, a copy of all tracks and sectors)
- creating a program that reads foreign disks and converts these to CP/M format

For all other normal programming you should use BODS calls.

One other note. The CP/M User Group has already discovered that many programs that were written under CP/M 1.3 using direct I/O must be converted before they will run under 1.4 or higher.

Table 6.3. Alternative I/O with CP/M

Using BIOS Calls	Using Direct Calls
Portable–can be run on any CP/M system.	Nonportable–often runs only on the CP/M version for which it was compiled unless patched.
No control over control codes, but upward compatible with all CP/M systems.	Control over use of control codes and isolation from CP/M's use of them.
All registers must be saved before calls.	Only registers you intend to use must be saved.
Logical I/O–CP/M controls the logical-to-physical mapping.	Hard I/O–you specify the specific device or disk track and sector you intend to use.

Used in most application software and as this method is more "portable"—the software can be used easily on other CP/M systems.	Used in programs that directly modify CP/M or address specific tracks and sectors (as certain types of copiers).

6.5 DISK CALLS

CP/M 1.4 has sixteen different disk functions available through BDOS calls (see Table 6.4). CP/M 2.X added seven additional disk function calls (see Table 6.5), and MP/M added another four (see Chapter 10). The first three of the new CP/M 2.X codes exist in CP/M 1.4, but are undocumented in the 1.4 literature.

Table 6.4. CP/M 1.4 Disk Functions

(Note: Function Code is placed in register C before call.)

Function Number	Function Description	Entry Parameter (in Registers D and E)	Returned Value in Registers A or A&B
12	Lift Head	(None)	
13	Initialize	(None)	(None) Disk A is logged in. Buffer set to 80H.
14	Select and Log In Disk	0 = A Disk, 1 = B Disk, etc.	(None)
15	Open File	FCB address	FCB setup. Byte address of FCB on disk returned—255H if not found.
16	Close File	FCB address	Byte address of FCB on disk if found, 255H otherwise. FCB written to disk.
17	Search for File	FCB address	Byte address on disk of FCB if found, 255H otherwise.
18	Search for Next Occurrence	(Same as 17)	(Same as 17)

19	Delete File	FCB address	E5H placed in ET field of directory FCB.
20	Read Record	FCB address (file must first be opened (15) or created (22))	Next 128 bytes (or logical sector) read to disk buffer in memory.
21	Write Record	(Same as 20)	Buffer written to disk sector, FCB CR incremented, A = 0 if successful.
22	Create File	FCB address. All data in memory FCB set to 0 except named type	Directory address of new file or 255H if directory is full.
23	Rename File	FCB address. Old file name in first 16 bytes, new file name in next 16 bytes	Directory address of old file or 255H if not found. Directory updated.
24	Interrogate Disk Log-In	None	HL returned with bits set for all drives on-line.
25	Interrogate Drive Number	None	Default disk # in A.
26	Set DMA Address	Address of 128-byte disk buffer	None. Subsequent read and writes will start from this address.
27	Interrogate Allocation	None	(Not used by application programs.)

Most CP/M software is upward compatible. That means that if a program runs on CP/M 1.4, it should also run on CP/M 2.X. (There are some exceptions, since some CP/M programs will not run on MP/M (see Chapter 10).) The converse, however, is not true. A CP/M 2.X program will not necessarily run on CP/M 1.4. The extra calls in the new CP/M permit random file access. Specified records within files can be read or written. Some programs, such as dBase II, sense

the version number of the CP/M and configure themselves for the specific version.

Table 6.5. Added Functions for CP/M 2.X

(Note: Function Code is placed in register C before call.)

Function Number	Function Description	Entry Parameters (in Registers D and E)	Returned Value in Registers A or A&B
28	Write Protect Disk	None	None.
29	Get Read/Only Vector	None	HL registers indicate drives which are R/O.
30	Set File Attributes	FCB address	Directory code.
31	Get Address of Disk Parameters	None	(Not used by application programs.)
32	Set/Get User Code	E = FFH (get), E = User code (set)	
33	Read Random	FCB address	A = error code, 0 if successful.
34	Write Random	FCB address	A = error code, 0 if successful.
35	Compute File Size	FCB address	(Used with #34 to append to end of file.)
36	Set Random Record	FCB address	Record # of position (used to build key indexes.)

(Note: For added MP/M functions see Chapter 10.)

In the early days of CP/M, some manufacturers developed their own operating systems that attempted compatibility with CP/M. In some cases the functions were copied only as far as function 27 (version 1.4), after which the manufacturer added his own function codes. Cromemco CDOS is an exam-

ple. It is only fully compatible as far as CP/M version 1.3. (If you are interested in Cromemco CDOS program conversion, see Jim Warner's article listed in Appendix E.)

When a file is opened by a program, CP/M allocates two areas in memory for that file. The first is the FCB or *file control block* and consists of 36 bytes (figure 6.3). This entry is very close to the directory entry and is, in fact, created from the directory entry when the file is opened (Function 15). Unless you specify otherwise, CP/M assumes this FCB buffer is located at 5CH (and within the first page of memory). This FCB is written back to the disk when the file is closed or when this extent is completed (16K) and another extent and directory entry is opened.

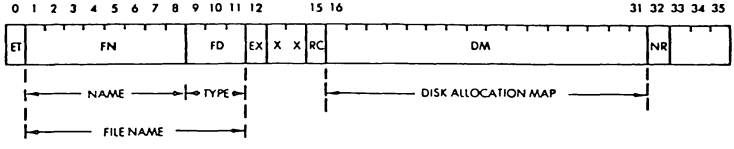


Figure 6.3. CP/M FCB Block

The other area of memory needed is a buffer area for the disk information. For standard systems and normal programming, this buffer holds enough for one sector (128 bytes) on a single-density system. Unless specified, CP/M defaults this buffer to start at 80H, again within the first page of memory. For multiple file processing or long buffers, the user will need to allocate space in the TPA for both the extra FCB's and the extra disk buffers. CP/M calls this starting address (normally 80H) the DMA address. This is misleading, since DMA normally means direct memory access. On the better CP/M systems, such as CompuPro, Digital Microsystem Computers, and some Tarbells, the transfer really is DMA. Other manufacturers do the transfer under program control. On *all* CP/M systems the BIOS listings and CP/M manuals will call the buffer start address the DMA address; but calling it one won't

make it one unless the software and hardware is specifically designed for DMA.

In Appendix I is an example listing of a MODEM program from the CP/M User Group written by Tom Pugh. This is a fairly simple CP/M program that shows some good examples of disk I/O processing. All of the concepts we have been discussing are illustrated there. Take the time to follow this program and see how the file is created (or opened), how the disk buffer is loaded, and how the buffer is written to the disk when full. Writing another version of this program for your own modem would be a good exercise in learning how the BDOS calls work.

Listing 6.1. Simple Modem Program

```
;PENNYWHILSTLE MODEM I/O
;Commands
;           control/D          end current mode
;           control/p          start or stop print
;Constants
fdos        equ     0005       ;entry to FDOS
clr         equ     01ah               ;clear screen
flag        equ     0h                 ;line feed flag
eot         equ     004h               ;end of xmission
cts         equ     003h               ;clear to send
rda         equ     001h               ;receive data available
cr          equ     00dh               ;carriage return
lf          equ     00ah               ;line feed
tbe         equ     10000000B                  ;tx buffer empty
dc3         equ     010h               ;control/P
fcb         equ     05ch
tfcb        equ     005ch              ;transient fcb
fcbnr       equ     fcb + 32
tbuff       equ     0080h
modem0      equ     0                  ;modem status port
modem1      equ     1                  ;modem data port
;fdos functions
rdcon       equ     1          ;read console character
wrcon       equ     2          ;write console character
wrlst       equ     5          ;write to list device
wrcbuf      equ     9          ;write buffer to console
```

```
const   equ     11          ;get console status

        org 100h
        jmp start
        ds 40
stack:  ds 2                ;stack pointer
bufptr: ds 1                ;data buffer ptr
prtflg  db 1
lfdflg: db 1

msgl:   db      clr,'Modem Driver',cr,'$'
msg3:   db      'End of connection',cr,'$'
msgl8:  db      'Waiting for model',cr,'$'
msg4:   db      'Connection established',cr,'$'

;support subroutines

mdm:    ret

stop:   mvi c,wrcbuf
        lxi d,msg3
        call fdos
        jmp restrt
exit:   xra a
        lhld stack
        sphl
        ret

tchar:  mvi c,wrcon
        mov e,a
        call fdos
        ret

cstat:  mvi c,const
        call fdos
        ora a
        ret

crd$st: mvi c,rdcon
        call fdos
        cpi eot
        rnz                 ; ↑D?
        pop psw             ; yes, fix stack—we won't rtn
        jmp stop

crlf:   mvi a,cr
        call tchar
        mvi a,lf
        call tchar
        ret
```

```
;main line program
start:  lxi h,0
        dad sp
        shld stack          ;save cp/m stack ptr
        lxi sp,stack
restrt: xra a
        sta lfdflg          ;reset line feed flag
        sta prtflg          ;reset print flag
        mvi c,wrcbuf
        lxi d,msgl
        call fdos
;wait for console input
        mvi c,rdcon
        call fdos
        cpi eot             ;control/d?
        jz exit
        mvi a,flag
        sta lfdflg                  ;set line feed flag
        mvi c,wrcbuf
        lxi d,msgl8
        call fdos
an0:    call cstat
        jz anl
        mvi c,rdcon
        call fdos
        cpi eot
        jnz anl             ;↑D from console?
        xra a
        jmp stop
an1:    mvi c,wrcbuf
        lxi d,msg4
        call fdos
trchar: in modem0
        ani rda
        jnz nochar          ;no, test for local character
        in modem1
        push psw
        call chkprt         ;check and echo on printer if
                            necessary
        call tchar          ;echo on console
        lda lfdflg
        ora a
        jz trl
        pop psw
```

```
            cpi cr
            jnz nochar
trlf:       mvi a,lf        ;send a line feed
            call tchar
tr0:        in modem0       ;busy outputting?
            ani tbe
            jnz tr0
            mvi a,lf
            ani 7fh
            out modem1      ;xmit
            jmp nochar
tr1:        pop psw
            jmp nochar
nochar:     call cstat
            jz trchar       ;no local character
            call crd$st     ;read , test for ↑D
            cpi dc3         ;control p?
            jz tglst        ;yes, toggle printer
            ani 7fh
            out modem1
            call chkprt
            cpi cr
            lda lfdflg
            ora a
            jnz trlf
            jmp trchar
chkprt:     push a
            lda prtflg
            cpi 1
            jz prtout
aftprt:     pop a
            ret
prtout:     pop a
            push a
            mov e,a
            mvi c,wrlst
            call fdos
            jmp aftprt
tglst:      lda prtflg
            xri 1           ; toggle a with exclusive OR
            sta prtflg
            jmp trchar

A>type
```

Chapter 7

Creating CP/M Systems

Sooner or later you will want to create a modified version of CP/M from your current version. You did this when you first purchased your system and needed a copy of the master disk. Later, you may want to add a modem or change the memory size. This means changing the BIOS. The process of creating a new version of CP/M is called *system generation.*

7.1 TYPES OF SYSTEM GENERATION

There are three types of system generation:

1. Copying the existing CP/M to another disk with no changes.
2. Modifying a CP/M for use with another hardware system. (This could be the same hardware system, but only slightly altered–for example, with more memory.)
3. Creating CP/M on a hardware system for which no CP/M currently exists.

The first type of generation involves the use of the SYSGEN function (see *How to Get Started With CP/M*, by Carl Townsend, dilithium Press, 1981.) Only a brief overview of this will be given here. The third type is beyond the scope of this book, but is discussed in the Digital Research manuals. You should not attempt this type of system generation without a considerable background of experience with CP/M. The second type of generation, however, can easily be done if you

have some level of CP/M experience. This second type of generation is the main subject of this chapter.

One word of warning. Many manufacturers have changed the way CP/M is generated for one reason or another. Some of these variations will be discussed later in the chapter. This chapter will still, however, give you a basic functional understanding of how CP/M works.

7.2 COPYING CP/M

Objective: To create a duplicate of the existing CP/M system on a second disk.

CP/M, as it exists on most 8-inch disks, resides on the first two tracks of the disk. To create a copy of CP/M on a second disk, you must copy these first two tracks (0 and 1) to the first two tracks of another disk. NOTHING ELSE is changed on the destination disk. Neither the file directory (track 2) nor files on the destination disk are altered or destroyed.

NEVER work with the master disk you received when you purchased CP/M—except to create the first copy. After the first copy is made, store the master in a safe plalce (preferably in a separate geographic location) and create additional copies from the first copy.

The program used to copy the first two tracks is called SYSGEN. This program, along with the CP/M, should be on the source disk.

To start the copy, the user starts the SYSGEN program:

```
A>SYSGEN
```

SYSGEN first asks for the source drive name. Load the source disk and enter the source drive name from the keyboard. This will load the first two tracks of the disk to the transient program area of the computer memory.

SYSGEN then asks for the destination drive name. Load the destination disk and enter the corresponding drive name. The CP/M in the computer memory is then copied to the destination disk, and your new CP/M has been created. The new disk contains copyrighted material and should be properly labeled as such and protected from unauthorized use and copying.

If the new disk is placed in the primary disk drive, it should boot just like any other CP/M disk with the appropriate sign-on message. SYSGEN does not care when or if disks are swapped. No warm boot is needed, since the directory is not accessed or used after SYSGEN is loaded.

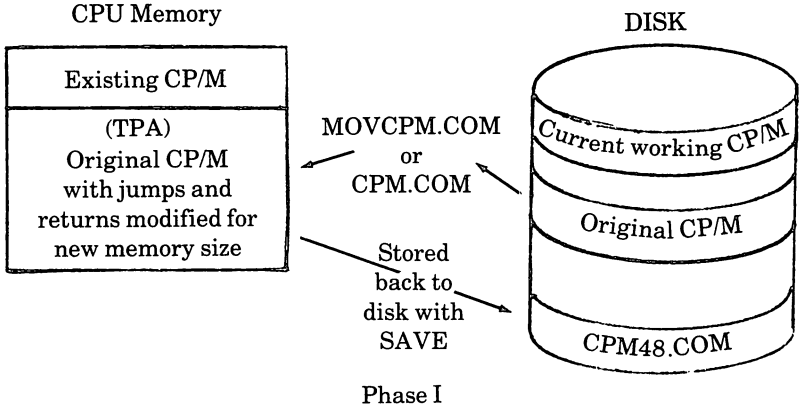
7.3 CHANGING AN EXISTING CP/M SYSTEM

Objective: To create a modified CP/M system from an existing CP/M system.

In generating our new CP/M system you will be working with four distinct copies or CP/M *images* (Figure 7.1). The role and definition of each should be clear.

1. The existing CP/M system that functions as the supervisor or manager of the conversion process.
2. A disk version of CP/M with the original BIOS that is created by CPM.COM (or MOVCPM.COM) and loaded to the TPA (transient program area) in computer memory with DDT.
3. A memory image of CP/M that is created and then altered in the TPA during the generation process.
4. The new CP/M that, after completion, is stored on the first two tracks of another disk.

NOTE: With some systems (for example, the Osborne I), the BIOS module is stored in read-only-memory. Such systems cannot be changed without reburning the PROM. For these systems, this section would not apply.



Phase I

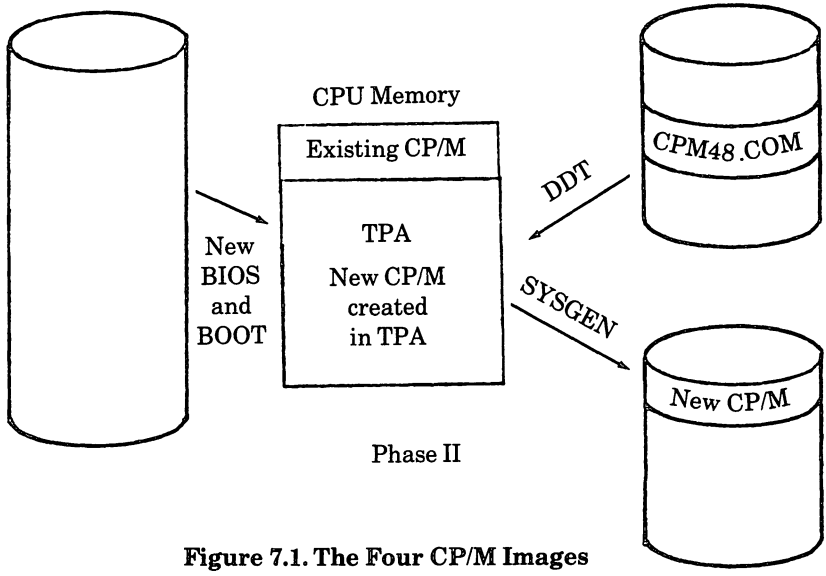


Figure 7.1. The Four CP/M Images

The basic generation process involves the following steps:

1. Create new BIOS.ASM and BOOT.ASM files that reflect the new physical driver, new memory size, and port assignments for the new system.
2. Assemble these using ASM or a macroassembler to a HEX form.
3. Use MOVCPM to create a CP/M image in the transient program area (Figure 7.1). This is NOT built from your working CP/M, but from the original BIOS using CPM.COM or MOVCPM.COM. It contains the jumps and returns to reflect the new memory size.
4. Save this image on disk with the SAVE command.
5. Using DDT, move this image again to the transient program area (Figure 7.2).
6. Using the "I" and "R" commands of DDT, overlay the memory-image of the original BIOS and BOOT with the new HEX files from the BIOS and BOOT you have created.

7. Use SYSGEN to store this new memory image to the first two tracks of a new disk.

The part to be entered by the operator is underlined (the asterisk is **not** entered by operator).

```
B>ed fbios16.asm <CR>
*100a <CR>
*f16 <CR>
*-1 <CR>
*
MSIZE   EQU   16                ;MEMORY SIZE IN KBYTES.
*s16↑Z48 <CR>
*-1 <CR>
* <CR>
MSIZE   EQU   48                ;MEMORY SIZE IN KBYTES.
*e <CR>
B>ed sboot16.asm
*#a <CR>
*f16 <CR>
*-1 <CR>
;                                                        **
* <CR>
MSIZE   EQU   16      ;MEMORY SIZE IN DECIMAL KB.        **
*s16↑Z48 <CR>
*e <CR>
B>asm fbios16 <CR>
CP/M ASSEMBLER-VER 1.4
BE2D
008H USE FACTOR
END OF ASSEMBLY
B>asm sboot16 <CR>
CP/M ASSEMBLER-VER 1.4
```

```
0080
001H USE FACTOR
END OF ASSEMBLY

B>cpm 47 * <CR>

CONSTRUCTING 47K CP/M VERS 1.4
READY FOR "SYSGEN" OR
"SAVE 32 CPM47.COM"
B>save 32 cpm47.com <CR>
B>ddt cpm47.com <CR>
DDT VERS 1.4
NEXT PC
2100 0100
-ifbios16.hex <CR>
-r6480 <CR>
NEXT PC
2298 0000
-isboot16.hex <CR>
-r900 <CR>
NEXT PC
2298 0000
-↑C
B>sysgen <CR>
SYSGEN VER 1.4
SOURCE DRIVE NAME (OR RETURN TO SKIP) <CR>
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B
DESTINATION ON B, THEN TYPE RETURN <CR>
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) <CR>
```

Figure 7.2. Generating a New System with a Tarbell Controller

7.4 OFFSET AND BIAS

In the creation of the new system, two terms are used which are very important. One of these is *negative offset*, the second is the *bias*. These can be defined using Figure 7.3:

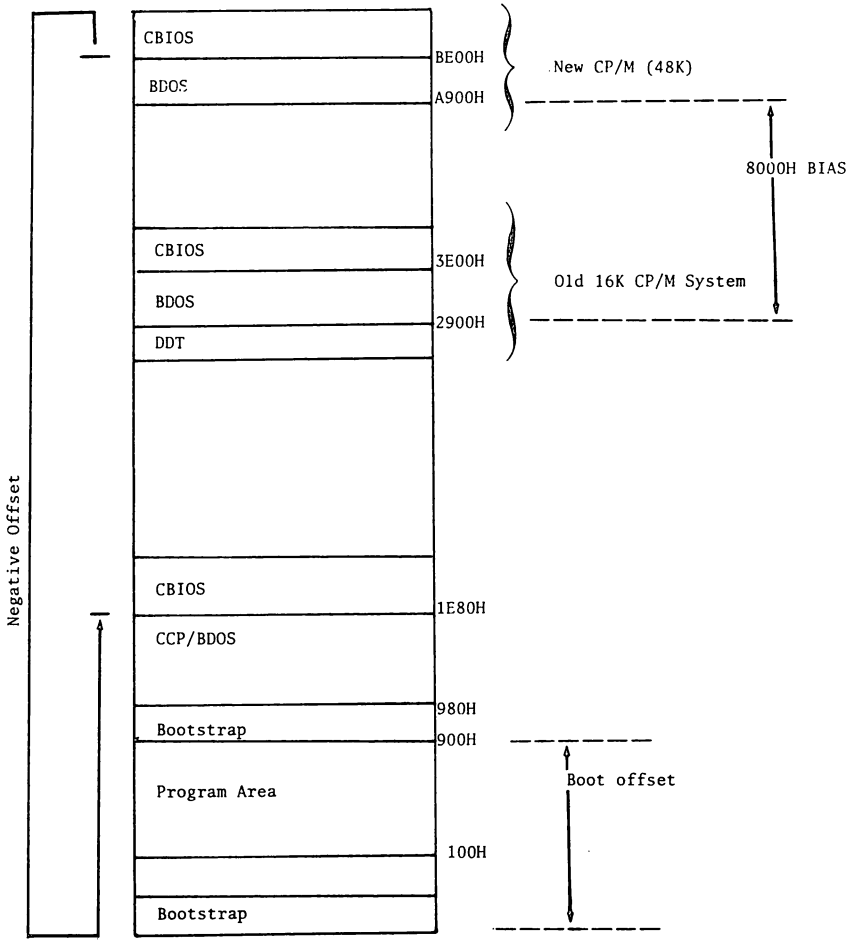


Figure 7.3 Generating a CP/M 1.4 48K System

negative offset–the distance from the BIOS image of the new system to the BIOS image in the old system at the top of the memory.

bias–the distance from the new CP/M CCP to the CCP of the distributed CP/M. Basically, this is the distance to move the distributed CP/M.

The negative offset is a bit tricky. Since DDT can only move a file forward (up) in memory, the BIOS must be moved up and over (the 64K boundary) to locate it in the TPA. The offset is the distance the BIOS must be moved.

Notice that while the BOOT can be moved upward to place it in the TPA, the BIOS moves out of memory if moved upward. DDT can only move an image upward. To resolve this we need to remember that the memory wraps around after 64K. Thus, the offset is really a negative number, and we express it as a positive number by calling it a negative offset.

The bias is the distance from the CCP of the CP/M system being built (at the low end of the TPA) to the CCP of the distributed version of CP/M assuming that it is residing in memory (Figure 7.3).

Some manufacturers call the negative offset a bias (as Tarbell does), whereas Digital Research (and this book) define the negative offset and the bias as different concepts. Also, offsets and the bias will vary between 1.4 and 2.2 systems. CP/M 1.4 is distributed as a 16K system, while the new CP/M 2.4 is a 20K system. The 20K system, when it is in memory, is further from the new CP/M being created, and therefore has a larger bias. The negative offset, however, is higher because the TPA BIOS is higher in memory and the old BIOS must be moved further "up and over."

7.5 THE CREATION

Before we start, you should be aware that some BIOS modules use more space than CP/M normally permits. The distributed version of CP/M allows up to 384 bytes in the 1.4 system, and up to 768 bytes in the 2.X system. The BIOS size can be expanded, however, by generating CP/M for a slightly smaller memory size. As the BIOS is at the top of the CP/M, moving CP/M down permits more room for the BIOS. For example, if you have 64K of memory and wish to generate a 1.4 system with an extra 1K of BIOS, generate the system for 63K. This will place the CP/M 1K lower in memory leaving an extra 1K at the top of CP/M for the BIOS.

As an example, we will generate a 1.4 48K CP/M system using the steps described earlier. First, generate a working disk with the following programs:

DDT.COM	(the dynamic debugger)
ASM.COM	(the assembler)
ED.COM	(the editor)

BIOS.ASM	(FBIOS, VBIOS, CBIOS, etc. will all do—it varies with the manufacturer)
BOOT.ASM	(ditto)
MOVCPM.COM	(also called CPM.COM on some systems) Creates CP/M for new memory sizes

Erase all unneeded programs on the disk, since you will need plenty of working room on the disk. Now armed with this disk, the CP/M manuals, your manufacturer's instructions, scratch paper, and a sandwich, you can disappear to your room with your computer system to work your magic. It is also helpful to know how to work addition and subtraction with hexadecimal numbers. You can use a calculator for this, but DDT does have a built-in hexadecimal calculator for those of us who were raised on the old math. Now, mount the work disk on drive A and boot it up.

NOTE: Remember that your current operating system is used to manage and supervise the generation process. In executing MOVCPM, the system compares its internal MOVCPM CP/M serial number with the operating system CP/M serial number. If the serial numbers do not match, a synchronization error message will be displayed. Always use the same MOVCPM that came with your distributed CP/M.

First, MOVCPM is called up with the desired memory size in kilobytes as a two digit parameter; in this case:

```
A>MOVCPM 48 *
```

Notes:

1. The asterisk is necessary.
2. If you are generating a 47K CP/M for a Tarbell 48K system, Tarbell calls MOVCPM "CPM" and generates to 1K less to give extra room for the BIOS as:

```
CPM 47 *
```

MOVCPM now moves a copy of CP/M to the TPA and changes all CALL and JMP instructions to agree with the requested memory size. This new CP/M is now in the TPA area starting at 900H as shown in Figure 7.3. We *could* use this CP/M and store it on tracks 0 and 1 of a disk in Drive B

with SYSGEN, but it wouldn't work. Both the BOOT and BIOS of this TPA system are the unmodified versions. We need to overlay these with our new versions of BOOT and BIOS. For the present, save the temporary CP/M using DDT on the disk as a file:

A>SAVE 32 CPM48.COM

Notes:

1. The 32 is the usual number for a 1.4 system. This indicates the number of 256 byte blocks we are saving to disk as a file. Thus the first 8K of the TPA is being saved as a file. Some manufacturers have larger requirements—if so, this will be mentioned in your system documentation.

2. If you are using a 2.X CP/M system, the value should be 34:

   A>SAVE 34 CPM48.COM

3. The file name could be different, but CPM48 communicates to you the size of the intended CP/M.

4. For a Tarbell with large BIOS you would want to use

   CPM47.COM

7.6 CALCULATING THE OFFSET AND BIAS

We now have to modify the BIOS module and BOOT for our personal CP/M system. To do this we need to calculate the bias and offset for the particular memory size we have planned. If you want to take the short cut in building your system, simply use the Table 7.1 and pick your bias and offset from the chart. Remember that if you plan to use a large BIOS (as with a Tarbell system), pick a memory size 1K less than your hadware memory. For example, to generate a 48K Tarbell system use the offset and bias values from the chart for 47K.

If you like to analyze what is happening and enjoy working with hexadecimal numbers, use Figure 7.3 to visualize how the offset is obtained. The computer can only address 64K of memory, but if you try to add one to the last addressable memory space (FFFF) you will get 0000. The high order bit is shifted left and lost. The distributed version of CP/M has a

BIOS that starts at 3E00H. The bias can be calculated for a 48K system as (the H indicates hexadecimal base):

BIAS = 48000 – 16000 = 32000 (or 8000H)

The new BIOS starting point can be found by adding 8000H to the BIOS start point of the 16K CP/M (as all CP/M locations are shifted by 8000H):

BIOS start location = 3E00H + 8000H = BE00H

To get the offset, we subtract the final BIOS start point from the TPA CP/M start point:

NEGATIVE OFFSET = 1E80H – BE00H = 6080H

The relationship between the negative offset and the bias can be expressed as:

negative offset = 980H – (2900H + bias) = E080 – BIAS (1.4 system)

negative offset = 980H – (3400H + BIAS) = D580 – BIAS (2.X system)

You can use DDT as a hexadecimal calculator for these calculations by using the H parameter as:

–H900,80

The computer will then display the sum and the difference of the two numbers.

Table 7.1. Offset Table

xx	bias	xx	bias	xx	bias
16	E080	19	D480	20	D080
23	C480	24	C080	27	B480
28	B080	31	A480	32	A080
39	8480	40	8080	47	6480
48	6080	55	4480	56	5080
61	2C80	62	2880	63	2480

7.7 ASSEMBLING THE BIOS AND BOOT

Using the editor, load the BOOT program. BOOT will load CP/M from the first two tracks of the disk to the desired final location in memory. You must specify, therefore, exactly where in memory to put it. There are two reasons for using this BOOT instead of fixed firmware on a chip. First, most of the intelligence required to load CP/M is now in software and only a small firmware chip is necesary to load BOOT. Second, the software BOOT is easily altered for different memory sizes or system upgrades.

First, locate the ORG statement by looking at the assembly code listing. This statement will have the form:

```
ORG XXXX
```

where XXXX is the address where BOOT will execute when it is loaded. For a Tarbell system this is 0000H. For a Thinker Toy system, this is 0080H. Record this number for your records—it will be needed later. Now examine the assembly code source listing and try to locate a memory size statement as:

```
MSIZE EQU 16
```

This is used by the assembler to calculate the loading address. Change this to 48K by typing:

```
MSIZE EQU 48
```

NOTE: Make this 48 regardless of what size BIOS you plan to use. If you cannot find this statement look for the group of statements:

```
LXI H,2900H
.
.
.
.
.
.
JZ 3E00h
```

and adds the bias (800H) to each of these numbers and edit to:

```
LXI H,A900H
.
.
.
.
.
.
JZ BE00H
```

Now save this file with control/Z and "e" and assemble it with:

A>asm boot

Now use the editor to call up the BIOS.ASM file and examine the first few locations. Look for the ORG statement and see if the BIOS starting point is controlled by an MSIZE statement such as:

MSIZE EQU 16

Change the 16 (or whatever number is listed) to the new memory size by typing:

MSIZE EQU 48

and make any other desired changes. If there is no MSIZE statement, look for the ORG statement and add the bias value to the value there:

ORG 3E00H (old)

becomes

ORG BE00H

Now, save the BIOS and assemble as:

A>asm bios

Now use DIR to check that the hex files for both BOOT and BIOS are on the disk. You now have two copies (ASM and BAK) of the large BIOS source and a large BIOS PRN file. Unless you have a dual-density disk, this has used most of the disk space. I normally erase the BIOS back-up file, as I keep the BIOS on several other disks. Later, after testing my new CP/M, I print and erase the PRN file.

7.8 THE GENERATION

The remainder of the procedure is very quick, since all the files needed are already created. First, move the new CP/M to the TPA with DDT:

```
A>DDT CPM48.COM
```

Now, move the load file of the BOOT to the TPA overlaying the distributed version of the CP/M BOOT:

- IBOOT.HEX
- R900 (if BOOTORG is 000H)

Note:

1. The R value is the offset for the BOOT and is 900 for the Tarbell system and 880H for a Thinker Toy system:

Now, load the BIOS into the TPA in the same way:

- IBIOS.HEX
- R6080

NOTES:

1. The R value here is the calculated negative offset or Table 7.1 value.
2. For a Tarbell system, the offset should be calculated for a 47K system giving an R value of 6480H.

Now, do a control/C to exit DDT. The new CP/M is still in the TPA.

Now, load an empty formatted disk in the right drive and call up SYSGEN. Use SYSGEN to move this TPA image to the first two tracks of the new disk (see Figure 7.1).

The generation will not destroy or alter any files already on the disk to which you loaded the CP/M. Any additional files you wish to add must be transferred from another disk with PIP.

7.9 SYSTEM VARIATIONS

Many system manufacturers have added their own variations to CP/M. Sometimes these were necessary for CP/M to work on a particular system; at other times, these are enhancements to the basic CP/M.

CP/M on Radio Shack and Heath systems must be modified, as some of the memory normally used by CP/M is read-only-memory. This necessitates a relocation of the starting point for CP/M programs. Other systems store the BIOS (or the entire CP/M) in read-only-memory. Such systems are not alterable without reburning the PROM. The Osborne computer also has a PROM, and uses a shadow or bank select method of switching this PROM in and out.

Systems Group stores the CP/M as a file on the disk much like the MP/M system strategy (see Chapter 10). Although this file takes space on every diskette, the loading time on this system is much, much faster than most other CP/M systems because of the DMA, interrupt processing, and extensive disk buffering.

Chapter 8

Database Management

The following hypothetical account is typical of the way many companies proceed toward more and more extensive computerization of their various business functions.

Widget Manufacturing Company decided to computerize their inventory system, accounting, and financial analysis. For several years inventory management had been done by a program written in BASIC, but now they wanted to add a purchase order system and integrate this with the accounting function. There were several accounting packages available, but these were written in another version of BASIC using a different file structure. Nothing could be integrated, and the backlog increased daily. In addition, management was requesting a wide variety of reports that could not be generated rapidly with BASIC. Programmers were busy maintaining the programs currently in use.

Widget Manufacturing decided that the solution was to establish and maintain all of its computer-readable data in a single pool or bank—a database—and design programs to extract or generate the desired information from this bank. A programming language for such an environment should be able to extract such information quickly and process it easily without regard to how it was stored. In other words, it should be possible to sort, index, and edit any of this information with a minimum of programming effort. Systems that are designed for this comprehensive type of information processing are called Database Management Systems, or DBMS.

8.1 DATABASE MANAGEMENT

A Database Management System (or DBMS) is essentially a collection of programs that allow you to work with a whole collection of information. This collection of information is called a *database*. Program applications for DBMS include general ledgers, accounts receivable, inventory control, inquiry and mailing list systems, order entry systems, program directories, quotation files, bibliographies—essentially almost any application in which random-access data files are processed. Sometimes the user can write the desired programs in BASIC or some other familiar high-level language. Calls are made from BASIC to routines in the DBMS. Indexing, sorting, and report routines are shared by all programs using the DBMS. Other DBMSs, however, are written in their own programming languages.

There are several advantages of a DBMS:

1. Prevention of Data Duplication

   Duplication of data occurs when a system grows to meet many needs and files are provided for each of these needs. Soon the same name and address or inventory part is appearing in several files and you discover that someone is typing in information for one program that was entered for another purpose in another program. With database management, information in a common data pool is shared by many programs.

2. Reduced Program Development Time

   Since the programmer does not have to maintain the files, programming time is greatly reduced. Some popular DBMS, such as dBase II, have a very high-level language with built-in structures to reduce programming time to near zero. Indexing, sorting, and report generation are all included. Other DBMSs, like KBASIC, use extensions to an existing language to simplify indexing and file management, but still permit all the standard BASIC features to be used.

3. Faster Programs

   Data access in a DBMS is with assembly-level programs which are generally faster than interpreted BASIC. Compiled BASIC programs using KBASIC are

extremely fast. dBase and other relational DBMSs are often slow when many files or array variables must be used. With one or two files, however, such relational systems are quite fast.

4. Increased Data Reliability

   As a typical system evolves and the number of files and the complexity of their interrelationships grows, sooner or later problems arise. Perhaps the system reuses an old part number when file space becomes short. Often the problem is undiscovered for weeks or months because of the complexity of the system. By then, it is almost impossible to rebuild the file. A DBMS makes it far easier to maintain the integrity of the data between programs and to monitor the data flow for errors and inaccuracies.

5. Preservation of Data Integrity

   In a DBMS certain data fields always contain certain types of data. Sex will always be "M" or "F." Zip codes are always numerical (unless you include Canadian zips). An account cannot be deleted if a balance is outstanding. All of these checks are made easier and faster using database management techniques.

6. Provisions for Security

   A great deal of the data in any database must be made secure to prevent unauthorized access. For example, an employee's salary should not be available to every user. This is relatively easy with floppy disks since the disk can be removed to locked file. Database systems that use hard disks, however, must have provision for data security written into the software. This is much easier in a DBMS using comprehensive file management techniques and a single, central data bank.

7. Flexibility

   Data used by programs in a DMBS is stored in a well-defined, well-planned structure. Modifications can easily be made for later software. For example, if you have an order entry and shipping system, you might want to add a credit limit field to the account master record. In a properly designed DBMS this addition and change is easy.

Let's look at two systems on the market and compare these to see the advantages and disadvantages of each. Most of the systems on the market are variations of one of these types.

8.2 KBASIC AND KEYED INDEXED SEQUENTIAL INDEXING

With most BASIC programs, you have a choice of two file accessing methods: sequential and random. Sequential files are used in applications in which the information must be serially read or written. An example would be a word processing program. To modify any part of the file, all of the file up to that point must be read. Clearly with files of any length, access time can be slowed considerably. A user with a word processor, however, generally works with linear information flow and the limitation of serial access is not a problem.

The second file type is random access. Random access files store information in blocks of fixed-length records. A particular record can be accessed directly, regardless of where it is stored in the file, as long as some type of index is maintained to indicate where the information is stored. For example, a mailing list program using random access files would use one or more indexes to locate any address entry for updating or printing. The entire index could be stored in the computer memory if it is short or, if it is long, it could be stored as a separate sequential or random file.

An obvious advantage would be to have some type of function built into the BASIC to minimize the programs having to manage this index. The programmer can then concentrate on managing information and leave the indexing to the program. Keyed Indexed Sequential Indexing provides just such facilities. The result is shorter programs that run much faster, and development time is greatly reduced.

One software product that permits keyed indexed sequential indexing is KBASIC. KBASIC is the well-proven Microsoft BASIC (version 4) with the addition of nine commands and one function. Using these features, you can index files with a minimum of programming. Typical applications include programs managing mailing lists, quotations, bibliographies, and resources against a variable number of keywords for each item. A version of KBASIC is also available for compiled BASIC (BASCOM) and Fortran.

The basic strategy is that you, as the user, generally want to access a record by certain keys or descriptors—not by its position in the file. If George Jones, for example, has a new box number for his mailing address, you want to locate George's address quickly without knowing or caring whether he is the first or the 200th address in the file. The KBASIC commands and functions allow you to do just that. The type and number of keys can be selected individually for each address. It could be George's name, his zip code, perhaps his city, or whatever special interests George has.

KBASIC stores information in two types of files in addition to the random and sequential files: the index and direct access data files. Random and sequential files are still available if these are needed. Direct access data files are defined with fixed record lengths of 64 to 512 bytes and store information in data blocks for each information entity (one address, one quote, or one resource). Index file records can be from 8 to 32 bytes in length. They contain the keys (or descriptors) and pointers to the associated data records.

Each index file record is composed of two parts. The first part is the key, and the second a data field. The key can be from 5 to 29 bytes in length, and the data field from 3 to 27 bytes in size. The total record size cannot exceed 32 bytes (see Figure 8.1).

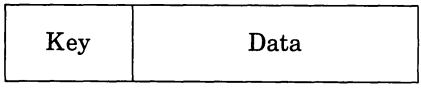


Figure 8.1. Index File Record

Normally, the data field is a pointer to the location of the data in the direct access data file, but in some applications with short data records it is permissible for the data itself to be stored in the data field of the index file; in this case no direct access data file is used. For most applications, it is quite adequate to use a three-byte data field in the index file, with the remainder of the record used to store the key. Of these three data bytes, one is the volume number of the data disk, and the remaining two bytes point to the record within the disk file.

KBASIC does not permit duplicate keys within the index file. Thus you cannot store an address that is already in the data file if the keys (the name for instance) are identical. This feature can be bypassed, however, using the FORCE function. For example, you might use a 16-byte index record with three bytes used as data and thirteen bytes used as the key. The user, however, only uses twelve of the thirteen bytes. KBASIC, using the FORCE function, can force a unique key from the twelve bytes by adding its own thirteenth byte. We will give a more specific example of this later, but for the moment it is only necessary to understand that you can decide whether duplicate keys are permitted in creating indexes.

The nine new BASIC commands available in KBASIC are shown in Table 8.1. Notice that some commands work only on data files, while others work only on index files. The special function PHONIC will be described later. In addition, KBASIC uses modified forms of the BASIC OPEN, CLOSE, GET, and PUT commands to manipulate the index and data files. With most of the commands the FIELD statement is used to create a buffer in a manner very similar to MBASIC. Thus, an index entry in the indexed sequential file is created using FIELD to place the descriptor and pointer in a buffer. We will give an example of this later.

Table 8.1. KBASIC Commands

Command	Index File Operation	Data File Operation
INIT	Initializes and creates an indexed sequential file.	Initializes and creates a direct access data file.
INSERT	Inserts the contents of the index file buffer to the file and returns a status code.	
RELESE	Removes the key in the index buffer from the index file and frees this memory space for reuse. Returns a status code.	Releases a direct access data file record for reuse. Returns a status code.

FORCE	Forces contents of index file buffer to index file and forces uniqueness of key. Returns a status code. Used to override duplicate names on file.	
SEEK	Searches the index file for the key in the index file buffer and loads the value of the direct access file from the index file to the buffer.	
ADVNC	Advances the index sequential file pointer one key and loads the index buffer with the key value and pointer to the direct access file. Use to scan the index file for an entry.	
UPDATE	Updates the pointer to the direct access file in an index file entry.	
STAT	Places the name of the last index file used in the index buffer.	Places status of last direct access file used in direct access file buffer.
ALLOC		Allocates one direct access data record for use by the index file.

8.3 A KBASIC EXAMPLE

Listing 8.1 shows a typical KBASIC program. This program maintains a keyword-indexed file of articles in a magazine publication. Entries can be easily added for each issue and the keyword-indexed listing printed. Notice how the index and data buffers are fielded and loaded in the same manner as with a random file. Note the high level of flexibility with KBASIC and the availability of all BASIC functions,

including array variables. With practice, you will build up a variety of subroutines that can be used in any program, making your programing job faster and easier.

The number of index files used depends on the application. For example, in a mailing list processor with three index files, the first index stores the keywords and names using a 12-byte key. The second stores the zip code in a 9-byte key, and the third stores the person's organization in a 24-byte key. As many as four indexes are permitted for a given file.

Listing 8.1 KBASIC Program to Index Magazine Articles

```
10 'PATTERNS INDEX PRODRAM
20 'Program: PATTRN
30 'Last Edit date: 7/6/80
40 'Author:Carl Townsend
50 CLEAR 2000
55 DIM KX$(5)
56 DIM KY$(30)
60 OPEN "X",1,"PAT",7,21,3,2
70 OPEN "D",2,"PAT",7,1,3,2
80 FIELD #1,21 AS K$,1 AS V$,2 AS LR$
90 FIELD #2,2 AS VO$,2 AS NO$,4 AS DT$,75 AS TI$,4 AS LL$,21 AS KX$(1),
21 AS KX$(2),21 AS KX$(3),21 AS KX$(4),21 AS KX$(5)
100 EN$="$"
120 LSET V$=MKI$(1)
130 PRINT"OPTIONS:"
140 PRINT"      0 to exit
150 PRINT"      1 to add entries
160 PRINT"      2 to print index on printer
170 PRINT"      3 to update entries
180 PRINT:INPUT"Option: ";P%
190 IF P%=0 THEN CLOSE:STOP
200 ON P% GOTO 400,1000,2200
210 GOTO 130
400 'add an entry
410 'load variables
420 LINEINPUT"Volume: ";F1$
425 IF F1$=EN$ THEN 130
430 LINEINPUT"Number: ";F2$
440 LINEINPUT"Date(xx/xx): ";F3$
```

```
430 LINEINPUT"Article title: ";F4$
460 IF F4$=EN$ THEN 420
465 INPUT"Length in pages: ";F5
470 J%=1
480 LINEINPUT"Keyword: ";KY$(J%)
490 IF KY$(J%)=EN$ THEN 530 'all keywords in
500 J%=J%+1
510 GOTO 480
520 'store variables in file
530 ALLOC 2,L%
540 FOR K%=1 TO J%-1
550 IF KY$(K%)="" THEN NEXT K%
560 LSET K$=KY$(K%)
570 LSET LR$=MKI$(L%)
580 FORCE 1,S%   'load key to index
590 IF S%=0 THEN NEXT K%:GOTO 630 'exit key storage from here
600 IF S%<0 THEN PRINT "FILE FULL":STOP
610 IF X%>0 THEN PRINT "KEYWORD OVERFLOW":RELESE 2,L%:STOP
630 LSET VO$=F1$
640 LSET NO$=F2$
650 LSET DT$=LEFT$(F3$,2)+RIGHT$(F3$,2)
660 LSET TI$=F4$
670 LSET LL$=MKS$(F5)
680 FOR M%=1 TO 5
690 LSET KX$(M%)=""
700 NEXT M%
710 FOR M%=1 TO J%-1
720 LSET KX$(M%)=KY$(M%)
730 NEXT M%
740 PUT 2,L%
750 GOTO 450
1000 GOSUB 2000 'print headings
1020 SEEK 1,S%
1040 ADVNCE1,S%
1050 IF S%<>0 THEN 130
1060 X$=LEFT$(K$,20)
1070 GET #2,CVI(LR$) 'get record
1080 F1$=VO$+"/"+NO$
1110 D$=LEFT$(DT$,2)+"/"+RIGHT$(DT$,2)
1130 IF X$<>Y$ THEN Z$=X$ ELSE Z$=SPACE$(20)
1135 IF X$<>Y$ THEN LPRINT:LP%=LP%+1
```

```
1136 T$=LEFT$(TI$,48)
1140 Q$=Z$+" "+F1$+" "+D$+"   "+T$
1160 LPRINT Q$;TAB(84);CVS(LL$)
1161 LP%=LP%+1
1165 IF LP%=>60 THEN LPRINT CHR$(12):LP%=0:Y$="":GOSUB 2100
1170 Y$=X$
1180 GOTO 1040
2000 LPRINT CHR$(27)+CHR$(26)+CHR$(73)  'reset
2005 FOR I%=1 TO 5:S=SQR(1):NEXT
2010 LPRINT SPACE$(7)+CHR$(27)+CHR$(57) 'set left margin
2015 FOR I=1 TO 50:S=SQR(1):NEXT
2020 LPRINT "              PATTERNS INDEX
2030 LPRINT
2040 LPRINT "Center For the Study of the Future
2050 LPRINT "4110 N.E. Alameda
2060 LPRINT "Portland, OR 97212
2070 LPRINT:LPRINT DA$:LPRINT:LPRINT
2080 LP%=9
2100 LPRINT "Descriptor";TAB(21);"Vo/No Date    Title";TAB(84);"Length"
2110 LPRINT:LP%=LP%+2:RETURN
2200 'update of entries
2210 PRINT "Update Options:
2220 PRINT"     0  to return to main menu
2230 PRINT "    1  to correct a keyword spelling
2240 PRINT "    2  to delete a data entry
2250 PRINT:INPUT"Option: ";P%
2260 IF P%=0 THEN CLOSE:GOTO 50
2270 ON P% GOTO 2300,2600
2290 'correct keyword spelling
2300 INPUT "Old key: ";KO$
2301 IF KO$=EN$ THEN 2210
2305 KO$=KO$+SPACE$(20)
2320 INPUT "New Key: ";KN$
2330 Z%=20
2340 LSET K$=KO$+SPACE$(20)
2350 SEEK 1,S%
2360 IF S%<0 THEN 2300
2370 IF S%>0 THEN PRINT "KEY NOT FOUND":GOTO 2300
2372 IF LEFT$(K$,Z%)<>LEFT$(KO$,Z%) THEN 2300
2374 GOTO 2410
2380 ADVNCE 1,S%
```

```
2390 IF S%<0 THEN 2300 'EOF
2400 IF LEFT$(K$,Z%)<>LEFT$(KO$,Z%) THEN 2300
2410 GET 2,CVI(LR$)
2415 KK$=LEFT$(KO$+SPACE$(21),21)
2416 PRINT "Data Item: ";TI$
2420 FOR I%=1 TO 5
2430 IF KX$(I%)<>KK$ THEN 2460
2436 LSET KX$(I%)=KN$
2440 PRINT "Data key re-spelled: ";KX$(I%)
2460 NEXT I%
2470 PUT #2,CVI(LR$)
2480 RELESE 1,S%
2490 LSET K$=LEFT$(KN$,20)
2492 FORCE 1,S%
2494 IF S%<>0 THEN STOP
2495 LSET K$=KO$+SPACE$(20)
2496 SEEK 1,S%
2497 IF S%<0 THEN 2300
2498 GOTO 2380
2600 'clears a data entry and all associated keys
2610 INPUT "KEY: ";KO$
2615 IF KO$=EN$ THEN 2210
2630 LSET K$=KO$
2634 Z%=20
2635 KO$=LEFT$(KO$+SPACE$(20),20)
2640 SEEK 1,S%
2645 IF S%<0 THEN STOP
2650 IF S%>0 THEN PRINT "KEY NOT FOUND":GOTO 2610
2655 GOTO 2680
2660 ADVNCE 1,S%
2670 IF S%<0 THEN 2610 'EOF
2680 IF LEFT$(K$,Z%)<>LEFT$(KO$,Z%) THEN 2610 'end of this key
2690 Q%=CVI(LR$)
2700 GET #2,Q%
2710 PRINT VO$+NO$+DT$+TI$
2712 FOR I%=1 TO 5:PRINT "KY :";KX$(I%):NEXT I%
2718 I$=""
2720 INPUT "Delete? ";I$
2730 IF LEFT$(I$,1)="n" THEN 2660
2740 IF LEFT$(I$,1)="N" THEN 2660
2750 IF I$="" THEN 2660
```

```
2755 RELESE 1,S%
2756 IF S%<>0 THEN STOP
2757 PRINT K$+" released"
2760 FOR I%=1 TO 5
2770 LSET K$=LEFT$(KX$(I%),20)
2775 PRINT K$
2780 SEEK 1,S%
2790 IF S%>0 THEN 2860
2800 IF S%<0 THEN STOP
2805 GOTO 2820
2810 ADVNCE 1,S%
2815 IF S%<0 THEN 2360
2820 IF CVI(LR$)<>0% THEN 2810
2830 PRINT K$+" released
2840 RELESE 1,LR%
2850 IF S%<>0 THEN STOP
2860 NEXT I%
2861 RELESE 2,0%
2862 CLOSE 1
2863 OPEN "X",1,"PAT",7,21,3,2
2864 LSET K$=K0$
2865 SEEK 1,S%
2866 IF S%>0 THEN 2610
2868 IF S%<0 THEN 2610
2870 RELESE 2,0%
2890 GOTO 2660
9000 'initialize
9010 KILL "PAT.DAF"
9020 KILL "PAT.NDX"
9030 INIT"X",1,"PAT",7,21,3,0
9040 INIT "D",1,"PAT",7,1,3,0
9050 RUN

OG>
```

8.4 THE PHONIC FUNCTION

The phonic function of KBASIC can be used to force a nine-byte code from a three-word name. The function ignores most titles. For example:

A$ = Mr. Tom L. Jones
B$ = phonic$(A$)

In this example a nine-byte code is created from A$ to form a key.

8.5 KBASIC SPEED

The access speed of KBASIC is remarkably fast. KBASIC achieves this by keeping a master index to the index in memory and using this to retrieve index pages as needed. The last index page used is always in memory, along with the master index page used. This means that a typical record access requires one, or at the most, two disk accesses (one for the data record and one for the index record). The access is practically immediate. Speeds for scanning a file using ADVNCE will be faster than you can read the output display.

8.6 DBASE II

dBase II is a relational DBMS that includes its own structured language. Any BASIC files you are currently using are considered "foreign" by dBase II and cannot be used in their present form. However, dBase II can read foreign files and use them to create dBase II files that can then be edited or printed just as any native dBase II file.

Program development time with dBase II is extremely fast, up to ten times faster than writing in BASIC. Indexing, sorting, and reporting can each generally be done with a single command. For example:

INDEX ON ZIP TO ZIP

will index a file on the field ZIP and store the index as ZIP.NDX. Whenever the file is updated, this index can be updated automatically. Powerful commands are included for formatting reports. In a typical application, even with little experience you can convert a "foreign file" (from a BASIC program) to a dBase II file and write all supporting programs (ADD, EDIT, DELETE, PRINT, etc.) in about half a day.

Programs are highly structured and, as such, are self-documenting. Listing 8.2 shows a typical program that main-

tains a disk directory. This file can be printed with heading, page numbers, date, and column headings, all with a single command using a stored report form.

dBase II 2.02 operates on only two files at a time, and this can be a limitation in some applications. Later releases will increase this to more than two. Also, the current version does not permit the use of array variables. This can make processing slower than interpreted BASIC if many files are accessed by keys, opened, closed, and scanned for a given application. Development time, however, is very fast and most programs run at excellent speeds. Indexing and sorting times vary greatly between different relational systems.

Table 8.2. dBase 2.03 Commands

? ⟨exp⟩ [,⟨exp⟩]
@ ⟨coordinates⟩ [SAY ⟨exp⟩ [USING '⟨picture⟩']] [GET ⟨variable⟩
    [PICTURE '⟨picture⟩']]
ACCEPT ["⟨cstring⟩"] TO ⟨memvar⟩
APPEND [FROM ⟨file⟩ [SDF] [DELIMITED] [FOR ⟨exp⟩]] or
    [BLANK]
BROWSE
CANCEL
CHANGE FIELD ⟨list⟩ [⟨scope⟩] [FOR ⟨exp⟩]
CLEAR [GETS]
CONTINUE
COPY TO ⟨file⟩ [⟨scope⟩] [FIELD ⟨list⟩] [FOR ⟨exp⟩] [SDF]
    [DELIMITED [WITH ⟨delimiter⟩]] or [STRUCTURE]
COUNT [⟨scope⟩] [FOR ⟨exp⟩] [TO ⟨memvar⟩]
CREATE [⟨filename⟩]
DELETE [⟨scope⟩] [FOR ⟨exp⟩]
DELETE FILE ⟨file⟩
DISPLAY [⟨scope⟩] [FOR ⟨exp⟩] [⟨exp list⟩] [OFF]
DISPLAY STRUCTURE
DISPLAY MEMORY
DISPLAY FILES [ON ⟨disk drive⟩] [LIKE ⟨skeleton⟩]
DO ⟨file⟩
DO WHILE ⟨exp⟩
EDIT
EJECT
ELSE
ENDDO
ENDIF
ERASE

```
FIND ⟨key⟩
GO or GOTO [RECORD], or [TOP], or [BOTTOM], ⟨n⟩
IF ⟨exp⟩
INDEX ON ⟨char string expression⟩ TO ⟨index file name⟩
INPUT ["⟨cstring⟩"] TO ⟨memvar⟩
INSERT [BEFORE], or [BLANK]
JOIN TO ⟨file⟩ FOR ⟨expression⟩ [FIELDS ⟨field list⟩]
LIST
LOCATE [⟨scope⟩] [FOR ⟨exp⟩]
LOOP
MODIFY STRUCTURE
MODIFY COMMAND ⟨command file⟩
NOTE or *
PACK
QUIT [TO ⟨list of CP/M level commands or .COM files⟩]
READ
RECALL [⟨scope⟩] [FOR ⟨exp⟩]
RELEASE [⟨memvar list⟩], or [ALL]
REMARK
RENAME ⟨current file name⟩ TO ⟨new file name⟩
REPLACE [⟨scope⟩] ⟨field⟩ WITH ⟨exp⟩ [AND ⟨field⟩ WITH
      ⟨exp⟩]
REPORT [⟨scope⟩] [FORM ⟨form file⟩] [TO PRINT] [FOR ⟨exp⟩]
RESET
RESTORE
RETURN
SAVE TO ⟨file⟩
SELECT [PRIMARY or SECONDARY]
SET ⟨parm⟩ [ON], or [OFF]
SET ALTERNATE TO ⟨file⟩
SET DEFAULT TO ⟨drive⟩
SET DATE TO ⟨string⟩
SET FORMAT TO ⟨format file name⟩
SET HEADING TO ⟨string⟩
SET INDEX TO ⟨index file⟩
SET MARGIN TO ⟨n⟩
SKIP ⟨ +/- ⟩ [⟨n⟩]
SORT ON ⟨field⟩ TO ⟨file⟩ [ASCENDING], or [DECENDING]
STORE ⟨exp⟩ TO ⟨memvar⟩
SUM ⟨field⟩ [⟨scope⟩] [TO ⟨memvar list⟩] [FOR ⟨exp⟩]
TOTAL TO ⟨file⟩ ON ⟨key variable⟩ [FIELDS ⟨field list⟩]
UPDATE FROM ⟨file⟩ ON ⟨key variable⟩ [ADD ⟨field list⟩]
      [REPLACE ⟨field list⟩]
USE ⟨file⟩ [INDEX ⟨index file name⟩]
WAIT [TO ⟨memvar⟩]
```

Table 8.3. dBase 2.03 Functions

@(⟨string1⟩,⟨string2⟩)	AT function
*	deleted record func
#	record number func
!(⟨char string⟩)	upper case function
$(⟨char string⟩,⟨start⟩,⟨length⟩)	substring function
⟨string1⟩$⟨string2⟩	substring search
CHR(⟨numeric expression⟩)	numeric to ASCII
DATE( )	system date func
EOF	end-of-file func
FILE(⟨file⟩)	existance func
INT(⟨numeric expression⟩)	integer function
LEN(⟨char string⟩)	length function
STR(⟨numeric expression⟩, ⟨width⟩[,⟨decimals⟩])	string func
VAL(⟨char string⟩)	value function
TRIM(⟨char string⟩)	trims strings
TYPE(⟨exp⟩)	supplies data type

Listing 8.2 dBasic Program to Maintain a Disk Directory

```
NOTE DIRECTROY ADD PROBRAM
* 6/22/81
ERASE
? 'ADD              DIRECTORY'
USE DISK INDEX DISKNO
DO WHILE T
        STORE '   ' TO MDISK
        @ 20,1  SAY 'DISK NUMBER ' GET MDISK PICTURE '999'
        READ
        IF VAL(MDISK)=0
                RETURN
        ENDIF
        STORE T TO FLAG
        DO WHILE FLAG
                FIND &MDISK
                IF #<>0
                        @ 21,1 SAY 'THAT DISK IS ALREADY ON FILE'
                        STORE 70 TO X
                        DO WHILE X>1
                                STORE X-1 TO X
                        ENDDO
                        STORE F TO FLAG
                ELSE
                        GO BOTTOM
                        STORE '           ' TO MCODE
                        STORE '                ' TO MDESC
```

```
                    @ 1,1 SAY 'DISK NUMBER '+MDISK
                    @ 2,1 SAY 'DESCRIPTION ' GET MDESC
                    @ 3,1 SAY 'SEARCH CODE ' GET MCODE
                    READ
                    APPEND BLANK
                    REPLACE DISKNO WITH MDISK,DESC WITH MDESC
                    REPLACE DESC WITH MDESC
                    replace code with !(mcode)
                    INDEX
                    STORE F TO FLAG
                ENDIF
            ENDDO
        ENDDO
        RETURN
```

8.7 LOOKING TO THE FUTURE

DBMS are definitely an important part of any microcomputer system that must process large amounts of randomly accessed data. Database systems using a very high-level language are rapidly gaining in popularity because of their ease of programming. The older hierarchial database systems still offer speed and flexibility advantages, as well as being capable of interfacing to existing languages.

For more information on database management systems for CP/M, see *CP/M Database Management Systems* by Carl Townsend (dilithium Press, 1983).

Chapter 9

Recovering Lost Programs

The most careful of us will occasionally erase a file by mistake. Although a directory or status listing will not show the erased file, it is often possible to recover it – particularly if you have not created any new files on the disk since the erasure.

To recover the file you will need either the DU program from the User Group (Volume 40 for CP/M 1.4, Volume 46 for CP/M 2.2) or any of the available "disk doctor" programs available commercially. For the examples here, we will use the DU program. You can use any program that can access the disk by track and sector and alter a given sector. Since most software of this kind uses direct BIOS calls, you must be sure it will work with your CP/M version.

Try this first with a back-up disk. Learn the principles and *then* apply it to recover that lost file.

Let's assume that the first part of your directory looks like this:

```
A>dir
A: TEXX     COM
A: DU       COM
A: XDIR     COM
A: MARGIN   COM
A: SBOOT24  COM
A: PRINT    COM
A: COPY     ASM
```

Now suppose that you erase a file accidentally:

```
A>ERA MARGIN.COM
```

A directory scan of the disk or a STAT will not list this file; therefore it cannot be accessed by the usual file commands:

```
A>dir
A: TEXX     COM
A: DU       COM
A: XDIR     COM
A: SBOOT24  COM
A: PRINT    COM
A: COPY     ASM
```

The program is still on the disk, however. The ERA command only changed two bytes in the directory record. By restoring these bytes we can recover the files—*if* nothing has been written over it.

Now call the DU command and the M command to get a map of the disk. For physical access purposes, the disk is considered to be divided into "groups" of 1024 bytes each. A group is 8 sectors. Group 0 is the first group in the directory on Track 2.

```
A>du
DISK UTILITY (DU) V3.0 01/07/80

Type ? for help
Type X to exit

:m
02-11   TEXX        .COM 0   :12-21   TEXX        .COM 1
22-23   TEXX        .COM 2   :24-26   DU          .COM 0
27-28   XDIR        .COM 0   :29-29   DU          .COM 0
2A-1A   (MARGIN     .COM 0)  :2B-2B   MARGIN      .ASM 0
2C-2D   MARGIN      .PRN 0   :2E-2E   MARGIN      .HEX 0
2F-43 + +FREE+ +             :44-46   SBOOT24     .ASM 0
47-47   PRINT       .COM 0   :48-50   COPY        .ASM 0
51-59   CPM         .COM 0   :5A-60   PIP         .COM 0
61-62   SUBMIT      .COM 0   :63-75 + +FREE+ +
76-76   FBIOS24     .HEX 0   :77-85 + +FREE+ +
86-86   FBIOS24     .HEX 0   :87-8B + +FREE+ +
8C-91   ED          .COM 0   :92-99   ASM         .COM 0
9A-9E   DDT         .COM 0   :9F-A0   LOAD        .COM 0
A1-A3 + +FREE+ +             :A4-A4   FBIOS24     .HEX 0
A5-A7   STAT        .COM 0   :A8-A8   SYSGEN      .COM 0
A9-B0   COPY        .COM 0   :B1-B1 + +FREE+ +
B2-B4   SBOOT24     .BAK 0   :B5-B9   SBOOT24     .PRN 0
BA-C0 + +FREE+ +             :C1-C8   CPM55       .COM 0
C9-D4   FBIOS24     .BAK 0   :D5-D9   DISK        .TEX 0
DA-DA   DISK        .BAK 0   :DB-DE   FBIOS24     .BAK 0
DF-E4   FBIOS24     .BAK 1   :E5-E5   SBOOT24     .HEX 0
```

```
E6-ED   FBIOS24        .BAK 1   :EE-F2   CPM          .      0
F3-FF + +FREE + +              :
G = 00:0, T = 02, S = 01, PS = 01
```

This map lists the files in group order on the disk. Free group areas on the disk are identified, and erased files are identified with parenthesis.

Now go to group 0 and dump this group:

```
:g0
G = 00:0, T = 02, S = 01, PS = 01

:du_
?
:d
00  00544558  58202020  20434F4D  00000080  *.TEXX    COM. . . . *
10  02030405  06070809  0A0B0C0D  0E0F1011  *. . . . . . . . . . . . . . . . *
20  00544558  58202020  20434F4D  01000080  *.TEXX    COM. . . . *
30  12131415  16171819  1A1B1C1D  1E1F2021  *. . . . . . . . . . . . . . . ! *
40  00544558  58202020  20434F4D  0200000B  *.TEXX    COMM. . . . *
50  22230000  00000000  00000000  00000000  *"#. . . . . . . . . . . . . . . *
60  00445520  20202020  20434F4D  00000020  *.DU      COM.  .  .  *
70  24252629  00000000  00000000  00000000  *$%$). . . . . . . . . . . . . . *
```

Notice the directory is in this location—track 2, sector 1. Now step in one sector and dump this group:

```
:+1
G = 00:1, T = 02, S = 02, PS = 07

:d
00  00584449  52202020  20434f4d  0000000C  *.XDIR    COM. . . . *
10  27280000  00000000  00000000  00000000  *'(. . . . . . . . . . . . . . . *
20  E54D4152  47494E20  20434F4D  00000001  *.MARGIN  COM. . . . *
30  2A000000  00000000  00000000  00000000  **. . . . . . . . . . . . . . . *
40  0053424F  4F543234  2041534D  000E0018  *.SBOOT24 ASM. . . . *
50  44454600  00000000  00000000  00000000  *DEF. . . . . . . . . . . . . . . *
60  00505249  4E542020  20434F4D  00000004  *.PRINT   COM. . . . *
70  47000000  00000000  00000000  00000000  *G. . . . . . . . . . . . . . . *
```

We can now see the erased file entry in the directory. Notice that the only difference between the directory listing for the erased file and the entry for an active file is the first byte. The erased file entry starts with E5. The entries for the active files start with 00. If nothing has been written to the disk changing the E5 to a 00 will recover the file. To modify this entry, enter:

```
:ch20,0
E5
:d
00  00584449  52202020  20434F4D  0000000C  *.XDIR    COM. . . . *
10  27280000  00000000  00000000  00000000  *'(. . . . . . . . . . . . . . . *
```

```
20  004D4152 47494E20 20434F4D 00000001 *.MARGIN COM. . . . *
30  2A000000 00000000 00000000 00000000 **. . . . . *
40  0053424F 4F543234 2041534D 000E0018 *.SBOOT24 ASM. . . . *
50  44454600 00000000 00000000 00000000 *DEF. . . . . *
60  00505249 4E542020 20434F4D 00000004 *.PRINT COM. . . . *
70  47000000 00000000 00000000 00000000 *G. . . . . *

:w

:X
```

The ch instruction changes (c) the hex (h) location 20 (20) to a 0 (0). The old value E5 is printed by the command execution. A new dump shows the altered value. The "W" commands writes the sector to the disk and the "X" command exits the DU.

Now, if we do a directory listing we will see our file again.

```
A>dir
A: TEXX     COM
A: DU       COM
A: XDIR     COM
A: MARGIN   COM
A: SBOOT24  ASM
A: PRINT    COM
A: COPY     ASM
```

The file is intact and can now be accessed as usual.

WARNING: there is no guarantee this will work with your version of CP/M. Try this on a test disk first until you understand the principles involved. Then use it on real data with discretion.

9.1 RECOVERING PROGRAMS LOST DURING EXECUTION

A common frustration in running BASIC programs is seeing CP/M exit BASIC when you have not saved the program you are running. This generally happens because you have forgotten to insert a data disk or when the program detects a disk error in reading or writing a data file. Fortunately, if nothing else has been loaded to the transient program area, there is a simple way to get back into BASIC with your program still intact.

When the system exits BASIC, you will find yourself back at the operating system command level. Immediately create a file with no space assignment on the disk. For example:

```
A>SAVE 0 GO.COM
```

Then execute this dummy program (which doesn't exist):

```
A>GO
```

You will find yourself immediately back in BASIC with your program intact!

This is simply a way to trick the operating system. Acting as if a program has loaded, CP/M immediately goes to 100H and begins executing whatever was already there. SAVE is a built-in command, and executing it does not load anything to the TPA.

This could also be used to repeatedly execute the same program from the disk. In the above case, for example, BASIC would not reload from the disk. This means if you leave a given program (PIP, a formatter, or the editor) for any reason you can quickly get back by executing any dummy program with a zero length.

Although this dummy program can always be created on-the-fly, another solution is to use the disk name as a zero length file on all your disks. Executing the disk name, then, will always return you immediately to the last program executed—i.e., no program is loaded and the CP/M transfers to location 100H.

9.2 RANDOM NUMBER GENERATION

Most BASICs use a random number generator that does not create truly random numbers. You must give the generator a seed (starting value) before starting the program, and the same seed always produces the same series of random numbers.

There are several alternatives you can use to get a series of truly random numbers. One solution, however, that can be used with any CP/M system involves changing the BIOS.

First, modify the CONST and CONIN routine to run a counter any time the computer is waiting for input. For example:

```
CONST:  LDA   RANDO
        INR   A
        STA   RANDO
        IN    CSTAT
CONIN:  LDA   RANDO
        INR   A
        STA   RANDO
        IN    CSTAT
RANDO:  DS    1
```

Now, generate your new CP/M system with this BIOS. Scan the PRN file of the BIOS to locate the location of RANDO. This will be your random number "seeder."

Now, fire up BASIC and look at this location:

```
print peek(&HDB23)
```

Then use this to seed the random number generator in MBASIC using the following:

```
x = peek(&HDB23)
randomize (X)
```

9.3 AUTOMATIC STARTUP

Occasionally you may wish to create a CP/M system that will automatically execute a program on startup. This could be a program to set the clock, scan for appointments in a calendar, or simply print a sign-on message. This can easily be done in CP/M 1.4 and, with a little extra work, in later versions. CP/M always checks the place it stores the command line on boot-up to see if a command is present. By preloading this location you can create a single command line that will immediately execute on startup. To do this:

1. Count the characters in your command line including parameters and spaces between the command and its parameters. Do not count the carriage return. Example: SUBMIT TEST = 11 characters.
2. Convert each command line character to its hexadecimal equivalent.

3. Start the system generation, store the system image as you normally do with CPM 55 * (or whatever memory size you need), and save the resulting system (SAVE 32 CPM55.COM in this case).
4. Using DDT as in a normal generation, load the system created for the desired memory size.
5. Examine from location 0980H onwards to locate the Digital Research copyright and look backward from this to the first *address* ending in 00 or 80. This is the start of CCP, and is normally 0980 or 0A00 (see Figure 9.1).
6. Go in exactly seven bytes (0987 or 0A07) and overwrite with the number of characters in your command.
7. Starting with the next byte, overwrite with the hexadecimal characters of your command, overwriting a 00 after the last character.
8. Finish the CP/M patch as in a normal system creation by loading the BIOS and boot with the system and creating the new CP/M system on another disk.
9. Reboot this new disk and check to be sure the program executes.

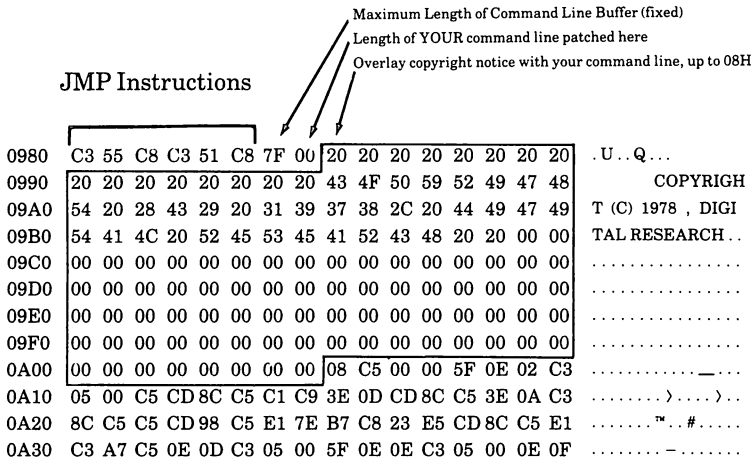


Figure 9.1. Creating an Automatic Startup System

Chapter 10

Multi-User and Networking Systems

In many applications it is necessary for more than one person to use the computer at the same time. MP/M was developed by Digital Research specifically for use in multi-user environments. MP/M is an abbreviation for Multi-Processing Monitor Control Program. This operating system permits a single processor (8080, Z80, etc.) to be shared by several users. although the processor is not *actually* being used by all users simultaneously, it operates fast enough that, by giving each user a small portion of time, each user thinks he has the system to himself. In fact, in some cases, you can run two programs of your own "simultaneously." MP/M is, therefore, capable of multi-tasking, multi-programming, and multi-terminal access.

Additional hardware is required for a multi-user environment. First, each user has a separate console with an I/O port (and supporting interface) for the console. In most cases additional memory is needed for each user. If you are shopping for a single-user system and plan to expand later to a multi-user system, the memory, processor, and disk controller should all use DMA and be interrupt driven. The processor and memory should have bank select capability.

The multi-user environment places additional demands on the operating system. For a microcomputer system, these demands are:

1. Program scheduling (the system must determine who runs what program and at what priority).

2. Synchronization of the various processes (file handling, printing, etc.).
3. Protection of files and processes running "simultaneously."
4. Memory management.

This requires a much more complex operating system than required by a single-user system. CP/M software is generally compatible with MP/M; however, there are some exceptions. If you are using MP/M and purchasing CP/M software, you should always check with the supplier for compatibility.

10.1 MP/M COMPONENTS

Like CP/M, MP/M consists of three components: CCP, BDOS, and the BIOS. MP/M can be configured in one of two ways:

A. Partitioned–Each user has a slice of a single 64K memory. This will only work if no user needs very much memory (for example, for certain control applications or perhaps a single user and modem interface, where the modem interface is a short program).

B. Bank Select–Each user has his or her memory bank. The processor switches between banks executing the program that has priority at a given time.

For most applications the bank select configuration is used. The major disadvantage is cost, as each user must be supported with memory. Normally each user would have about 48K of memory, with the top 16K of memory (to 64K) used by MP/M and shared by all users. Some memory cards currently on the market do not have the bank select capability. To use the bank select option, this is required. Bank select is the preferred configuration.

All of the utilities and programs included with MP/M exist as .PRL files. The old CP/M favorites, such as ED and PIP are there, but as PRL files. This means they are *relocatable*. This enables them to be used in systems without bank select by partitioning them in memory and starting from a location other than the standard CP/M 100H. With bank select, this

capability is not needed and COM files can be executed along with the PRL files. In fact, with the bank select installed you can use the CP/M editor (COM file) or execute programs purchased as COM files without any changes.

One word of warning. Some CP/M application programs may call CP/M system utilities that on the MP/M system do not exist, function differently, or have a PRL secondary name. In this case the CP/M application program will have to be modified to omit troublesome calls or it cannot be used. An example is dBase II's QUIT TO command, which builds a SUBMIT file to execute CP/M commands from within dBase II. This command will not work properly under MP/M. (dBase II is a fine product and well worth the money in spite of this idiosynchrasy.)

A memory map of a typical MP/M system is shown in Figure 10.1. This shows the memory allocation for a four-console system. The configuration and mapping of a typical MP/M system will vary. Unlike CP/M, there are several options to tune the system to your particular needs. The MP/M components are:

SYSTEM.DAT–A single memory page used by GENSYS (see generation) to control configuration. Used in operation to store flags.

CONSOLE.DAT–Console buffers. A 256-byte buffer is allocated for each console used.

USERSYS.STK–(Optional) Needed if you plan to run CP/M .COM programs.

XIOS–Hardware-dependent portion of the MP/M system (variant portion, similar to CP/M BIOS).

BDOS/ODOS–File and console manager. ODOS is a portion of BDOS that always remains in memory.

XDOS–MP/M nucleus and extended disk operating system. XDOS "extends" the functional capabilities of the BDOS to include the new MP/M functions.

RSP's–(Optional) These programs (ABORT, MPMSTAT, SPOOL, SCHED) can either be kept in memory all the time or loaded and run as needed.

BNKBDOS–(Optional) Needed if you are using bank select. Includes the software to manage the memory select.

MEMSEG.USER–These are user areas defined at generation. One exists for each memory bank. This is similar to the CP/M TPA. The first 100 bytes are used for user system parameters (FCB area, disk buffer, etc.). The remainder is available for programs. As with CP/M, on bank-select systems, programs start at 100H of the MEMSEG area.

TPA–An absolute memory area used by the system, based at 0000H.

	START LOC	LENGTH	
SYSTEM.DAT	FF00	0100	
CONSOLE.DAT	FB00	0400	BANK 0
USERSYS.STK	F900	0200	
XIOS.SPR	EB00	0E00	
BDOS.SPR	E300	0800	
XDOS.SPR	C400	1F00	
BNKBDOS.SPR	B600	0E00	
(USER AREAS)	0000	C000	(Bank Selected Memory Banks 1,2,3,7)

Figure 10.1. Memory Map of Four-User MP/M System

10.2 BANK SELECT

In a bank-select system, users have their own memory banks. When it is an individual user's turn, that user's bank is switched on by the processor and the program residing in that bank continues from where it was stopped. When that turn is up, the bank is switched off. The program is still in that memory bank, but it is dormant. In other words, all programs currently running at a given time are always in memory, each in its own memory bank. The processor can switch rapidly from one to the other as there is no disk loading necessary for the switch. In addition, if a single user is running more than one program, each program will be in a separate memory bank. There can be up to sixteen programs

running at one time, but each will need its own memory bank. In most systems the number of concurrent processes permitted at one time is determined by the number of memory banks installed in the hardware.

There are also other possible configurations. In the memory map of Figure 10.1, the disk drivers for a CDC 32-megabyte hard disk are kept in a separate memory bank. This makes it possible to fit the rest of the MP/M system with the buffers for four consoles into the upper 16K of computer memory (48K-64K).

10.3 CONSOLES AND USERS

When MP/M comes up, the prompt is slightly different from CP/M. The prompt includes a number and the letter designating the logged-on disk as:

```
0A>
```

The number is your *user number*; it is a logical assignment primarily used for partitioning programs and files on the disk. The number displayed is used with any file you access or write to determine a unique directory entry. With the exception of the PIP program, no user can access from or write to files of another user. The user number is stored with the file name in the disk directory and controls file access by other users. It can be used to partition the disk into logical segments that are assigned to different users. System programs are always User 0. Any user can change his user number by entering the USER command and the new user number as:

```
1A>USER 0
```

This changes the user number at the console to User 0.

NOTE: Unlike CP/M, there are no built-in commands in MP/M. All commands call programs from the disk. In this case, USER.PRL must be on disk A or an error message will be given.

Each console is also assigned a number by MP/M. This number is not displayed, but it can be found with the CONSOLE command. It cannot be changed by any user. The main console is console 0, and consoles are numbered sequentially as they are added. Several programs can be run at the same time from a given console.

When MP/M first comes up, the user number at each console is the same as the console number. The monitor console will display:

```
0A>
```

The next console will display:

```
1A>
```

If everybody is sharing files, the consoles should all be switched to User 0 with the USER 0 command. All users can use the User 0 area if desired.

The system files and any files created under USER 0 are accessible to anyone at any time. Thus USER 1 can access a USER 0 file, but if the file is rewritten it will then be a USER 1 file and no longer accessible to USER 0.

10.4 MP/M OPERATION

A multi-user system can be implemented in a variety of ways. MP/M uses a pooling method to service users; each user gets sequential time slices in a round-robin method. Scheduling is further complicated by I/O requests both to fast devices (such as the disk) and to slow devices (such as the printer). Figure 10.2 shows a flow diagram of a typical process. A typical program runs until an I/O request is made in a time-slice fashion with other programs. Once an I/O request is made, the program goes dormant until the I/O operation is completed. Then the program is awakened and the process is scheduled to run again at the next available time slice.

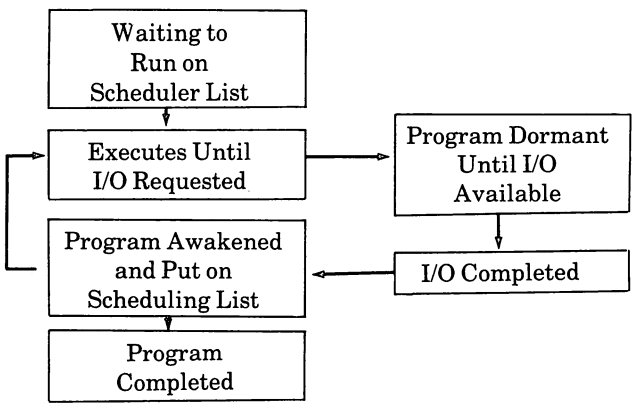

Waiting to Run on Scheduler List
Executes Until I/O Requested
Program Dormant Until I/O Available
Program Awakened and Put on Scheduling List
I/O Completed
Program Completed

Figure 10.2. Flow Diagram of an MP/M Program Execution

Notice that there are several "waiting lists." Typically, some jobs are waiting for the console, others for the disk, others for the printer, and others for a time slice. These waiting lists are called *queues*. The queue status can be displayed at any time with the MPMSTAT command. The queues are also managed by priority, with all jobs of the same priority serviced in a round-robin manner.

Devices can communicate their availability to MP/M using either a flat or vectored interrupt scheme. MP/M uses this hardware system to control a software interrupt system using flags. These flags, in turn, control job sequencing by giving MP/M the status on all I/O transfers.

The queues are managed much like CP/M files. They have FCBs, disk buffers, etc., and appear to MP/M as first-in-first-out (FIFO) files.

MP/M also includes a scheduler and spooler. You can spool a file to the printer, and it will be printed in order with other files spooled to the printer. The scheduler can be used to schedule jobs to run at a particular time if the system has a clock. This is useful for overnight back-ups.

10.5 MP/M COMMANDS

MP/M has all the standard CP/M commands, plus several new ones. In addition, some of the CP/M commands have additional features. MP/M has no built-in commands. It does contain the CP/M DIR, SAVE, ERA, REN, and TYPE commands, but they are files on the disk. They can only be executed from the disk on which they reside. You can also add commands as desired.

The new MP/M commands include:

ABORT	Abort the current process.
CONSOLE	Display console number.
DIRS	Display system directory.
DSKRESET	Change disks (similar to CP/M's warm start).
ERAQ	Erase, but query each file before erasing.
GENHEX	Create a HEX file from a COM file.
GENMOD	Create a PRL file from a HEX file.
SCHED	Schedule a job to run.
SPOOL	Schedule a print job to run.

STOPSPLR	Stop the spooler output.
TOD	Set or display time and day.

In addition, several new control characters have been added:

(Control) D	Detach console from current job (or attach console to the current job).
(Control) Q	Lock-out printer to other users.

Control/D is used to run several jobs from the same console. After you start a job, do a Control/D. The job will continue, but its console is now free for other tasks. You can now start another program from the same console. The first program runs until it needs the console (for user input, for example), then it stops and waits. When you do another Control/D, the console returns to the first job and performs any tasks needed by that program. The job then continues executing. This assumes, of course, that there are at least two memory banks available (one for each job).

Output to the printer is controlled with Control/Q. If two users use the printer at the same time the printer will print both outputs, garbling the printed text. To prevent this, a user can use Control/Q to lock the printer for a print process.

10.6 GENERATING MP/M SYSTEMS

The complete generation process is too lengthy to describe here, but it is summarized in Figure 10.3. MP/M is too lengthy to put on two tracks of a disk and is therefore loaded as file MPM.SYS on the disk. This is loaded and started with the program MPMLDR on the disk. If you have a disk with MPM.SYS and MPMLDR.COM on the disk, both configured for your system, you can start MP/M by entering MPMLDR.

MPMLDR is also normally stored on the first two tracks of the disk. A ROM boot can then load the two tracks and start MPMLDR, which in turn loads MPM.SYS, which starts MP/M (see Figure 10.3).

To create a new MP/M system, both MPMLDR.COM and MPM.SYS must be modified. Both have hardware-dependent portions. MPMLDR will be built from your current CP/M BIOS module. MPM.SYS, the MP/M system, has a hardware-dependent module called XIOS that is similar to the BIOS module. Only the general overview of the creation process is

given here. For details, consult Digital Research's manual and your system manufacturer's manual.

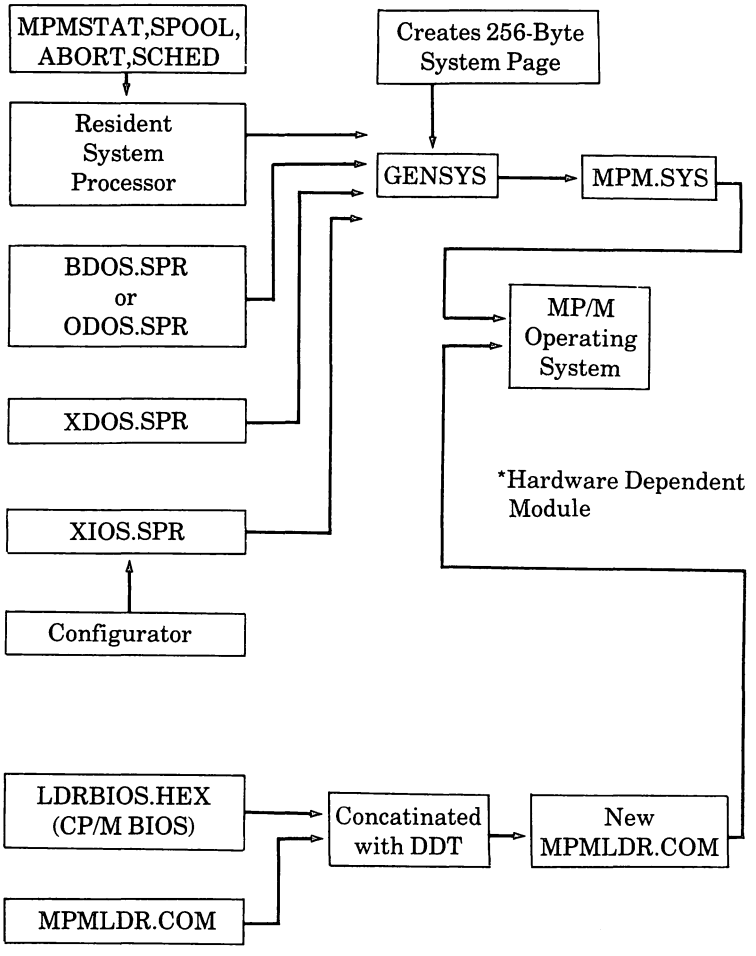


Figure 10.3. MP/M Generation Block Process

The several processes involved in MP/M generation can be grouped into two categories:

A. Creating MPMLDR to load the MP/M.

B. Creating MPM.SYS, the MP/M system.

MPMLDR is created under CP/M by creating a file LDRBIOS, which is the same as the CP/M BIOS you are currently using. This is concatenated with the MPMLDR that comes with the MP/M system. DDT or SID is used to combine these files. The composite image is saved on the disk with SAVE as the new MPMLDR.COM.

To create the new MPM.SYS, it is first necessary to create the XIOS.SPR module that is hardware dependent. This is created as an ASM file and assembled to a HEX file. An XIOS.SPR file is created from the HEX file with GENMOD. In some systems a configurator program will patch XIOS for the desired parts, eliminating the need to compile XIOS or use GENMOD. After XIOS.SPR is created, GENSYS is called. This queries you for system specifics to create the system data page. GENSYS then combines the BDOS, XDOS, XIOS, and any desired relocatable SPR modules to create the MPM.SYS file. This file can then be loaded and tested with MPMLDR.

10.7 DISTRIBUTED PROCESSING

MP/M uses a relatively small CPU to service up to sixteen users from multiple consoles. The key is "up to." This is fine for two users; but system response begins to deteriorate at four users and gets worse with each additional user. The tiny processor can only handle so much data.

Digital Research's solution was to provide a family of software systems to connect an MP/M system with other CP/M-like systems. Processing is then distributed over several processors with information on any system available to other systems. The control MP/M processor is the master, with one or more slave systems interconnected. The networking software is called CP/NET.

Chapter 11

Future Developments

The CP/M operating system is now well established with hundreds of support software and hardware manufacturers. Almost all serious microcomputer system manufacturers now support some versions of CP/M. Application software exists for almost every type of microcomputer application. As a result, CP/M will be around for a long, long time. Other operating systems exist with more enhancements and goodies, but none other has the wide support CP/M enjoys.

11.1 DATABASE MANAGEMENT

BDOS currently has no functions that support database management and ISAM files. The 2.2 release added random files to CP/M, but ISAM must still be implemented with costly database management packages that are three to ten times the cost of the basic operating system. At present there is no indication that such ISAM calls will be implemented. In addition, the extra memory required for such BDOS calls would erode the TPA size so much that many existing application programs (such as dBASE) would no longer run.

11.2 NETWORKING

If you've used MP/M with four or more users, you can understand the difficulty of trying to get good processing time in a single eight-bit processor chip. CP/NET is a major step in this direction, but there are still many development possibilities that are only visions and dreams at the current

time. I have no doubt that this will be a major expansion area for Digital Research.

11.3 CONCURRENCY

Concurrency is the capability of running two or more programs on the same system sharing a common processor but with separate and distinct memory areas. The console and keyboard are switched between the different jobs, which all appear to be running at the same time. This type of processing is a major area of development in the near future and will be very important to many users.

11.4 CONCLUSION

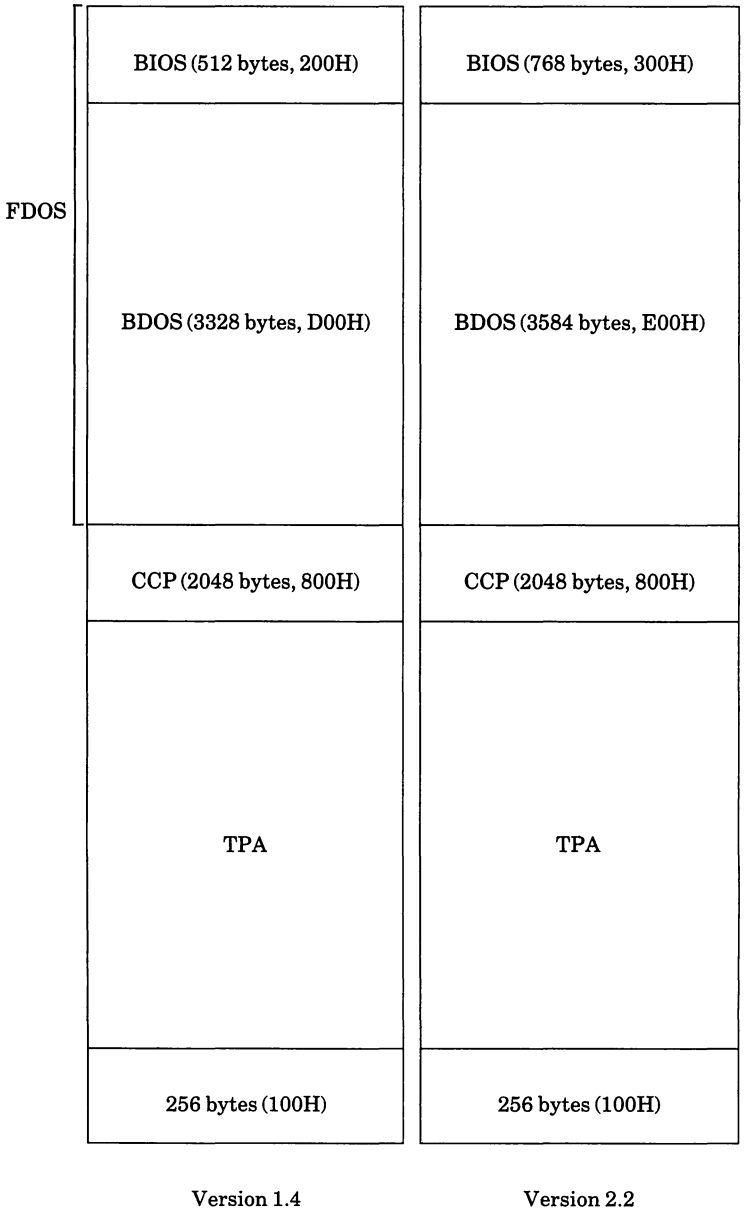
Local networking systems (or LANs in networking jargon) will quadruple in millions of line terminations between 1985 and 1990. Small interconnected systems will be an important systems concept for many businesses. The entry level cost for such data processing systems is affordable by almost any small business. Even large businesses will use these networks in addition to their centralized facilities. This system concept known as distributed processing will be an important concept in the late eighties. Digital Research will definitely attempt to reach this market with a variety of CP/NET-type systems.

CP/M, MP/M and CP/NET will continue to evolve as formidable products through the eighties. Later versions should always be compatible with earlier versions. This means application programs currently being used on these systems should be compatible with later versions. The primary exceptions to this rule would be programs using direct BIOS calls. In addition, the knowledge you have gained from this book (and other CP/M books) should be applicable in any future release.

Appendix A

Memory Maps

CP/M Memory Maps



First Page Memory Map

Location (Hex)

0-2	Jump to warm start entry in BIOS
3	Iobyte, if used
4	Default drive number
5-7	Jump to low end of BDOS (defines end of TPA if CCP is overlayed)
8-37	Reserved for user interrupt control
38-3A	Jump used by DDT in breakpoint mode
3B-5B	Not currently used
5C-7C	File information for files loaded to transient program area
7D-7F	Not used
80-FF	Disk buffer (128 bytes)

CP/M Disk Map (IBM Format)

Track #	Sector #	CP/M Module
00	01	Cold Start Loader
01	01-17	BDOS
01	18-26	BIOS (all of this track is not used)
02	01-26	Directory
03-76		Files, Data

Appendix B

Command Tables

CP/M Editor Command Summary

nA Append n lines from disk file.
+B Set character pointer to start or end of buffer.
nC Move character pointer n positions.
nD Delete n characters.
E Close and end edit.
nF Find nth occurrence (memory buffer only).
H Close and re-open files.
I Insert characters.
nJ Juxtapose strings.
nK Kill n lines.
nL Move character pointer n lines.
nM Execute macro n times.
nN Find (entire file) nth occurrence of string.
O Return to original file.
nP Display next 20 lines n times.
Q Quit with no file changes.
R Read library or stored lines from disk.
nS Change string.
nT Type or display n lines.
U Switch lower case to upper case.
V Switch to line number mode.
nW Write n lines to output file.
nX Store n lines in disk file.

CP/M Control Code Listing

Rubout Delete and echo last character typed.
Control/U Delete the entire console line.
Control/X Delete the entire console line.
Control/R Display current command line.
Control/E Local carriage return; no command issued.
Control/C Create a warm boot (BIOS is not reloaded).
Control/Z (Special to editor and PIP)
Control/P Copy all console output to printer.
Control/S Freeze console temporarily.

CP/M Commands and Utilities

Command	Function	CP/M 1.4	CP/M 2.2	MP/M
ABORT	Terminate a running program	no	no	yes
ASM	Assemble a program	yes	yes	yes
ATTACH	Attaches a detached program to console	no	no	yes
CONSOLE	Display console number	no	no	yes
DDT	Dynamic debugger	yes	yes	yes

DIR	Display file directory	yes	yes	yes
DSKRESET	Resets disks in multi-user system	no	no	yes
DUMP	Dump file to terminal	yes	yes	yes
ED	Editor	yes	yes	yes
ERA	Erase one or more files	yes	yes	yes
ERAQ	Erase files selectively	no	no	yes
GENHEX	Creates a HEX file from a COM	no	no	yes
GENMOD	Create a relocatable file	no	no	yes
GENSYS	Create an MP/M system from CP/M	yes	yes	yes
LOAD	Convert a HEX file to COM	yes	yes	yes
MOVCPM	Reconfigure CP/M to another memory size			
MPMLDR	Load and execute an MP/M	no	no	yes
MPMSTAT	Display MP/M system status	no	no	yes
PIP	Peripheral Interchange Program	yes	yes	yes
PRLCOM	Change a PRL file to COM	no	no	yes
REN	Rename a file	yes	yes	yes
SAVE	Save memory to a disk file	yes	yes	yes
SCHED	Schedule a program for execution	no	no	yes
SPOOL	Send file(s) to spool queue	no	no	yes
SUBMIT	Execute a program batch	yes	yes	yes
SYSGEN	Create a copy of CP/M	yes	yes	yes
TOD	Display or set time of day	no	no	yes
TYPE	Display file on console	yes	yes	yes
USER	Change USER area	no	no	yes
XSUB	Expand SUBMIT facility	no	yes	yes

CP/M PIP Command Summary

B	Block mode transfer
Dn	Terminate at n columns
E	Echo transfer to console
F	Remove form feeds from file
H	Hex check on transfer
I	Ignore :00 records in transfer
L	Translate upper case to lower
N	Add line numbers
O	Object file transfer
Pn	Include page ejects at every n lines
Qs↑z	Quit copying when string s is found
Ss↑z	Start copying when string s is found
Tn	Expand tabs to n spaces
U	Translate lower case to upper
V	Verify transfer
Z	Zero parity bits

(The ↑ means hold the control key down while pressing the next key shown.)

DDT Command Summary (all numbers in hexadecimal)

Axxxx	Enter assembly source statements beginning at xxxx.
D	Display the next 192 bytes of memory.
Dxxxx,yyyy	Display from xxxx to yyyy.
Fxxxx,yyyy,zz	Fill memory with zz from xxxx to yyyy.
G	Begin execution at the address in the program counter.
Gxxxx	Begin execution at xxxx.
Gxxxx,yyyy	Set breakpoint at yyyy and start execution at yyyy.
Gxxxx,yyyy,zzzz	Set breakpoints at yyyy and zzzz and start execution at xxxx.
I file name.typ	Set up FCB for file filename.type (see R command).
L	Disassemble and list next eleven lines of code.
Lxxxx	List and disassemble eleven lines from xxxx.
Lxxxx,yyyy	List and disassemble from xxxx to yyyy.
Mxxxx,yyyy,zzzz	Move memory contents from xxxx to yyyy to memory starting at zzzz.
R	Read a file to memory (see I command–which must be issued first).
Rxxxx	Read a file to memory offset xxxx bytes higher than where it normally loads (I command must be issued first).
Sxxxx	Display at xxxx and, optionally, alter the contents.
Txxxx	Trace xxxx steps of program (xxxx in hex!).
Vxxxx	Execute xxxx steps without tracing and stop.
X	Display CPU register contents.
Xr	Display register r and, optionally, change it.

Logical Names

CON: system CONsole
PTR: Paper Tape Reader
PUN: paper tape PUNch
LST: LiST device (printer)

Physical Device Names

TTY: TeleTYpe device
CRT: Cathode Ray Tube

BAT: console is reader (RDR) output to current LST: device
UC1: User defined Console #1
PTR: high speed Paper Tape Reader
UR1: User defined Reader #1
UR2: User defined Reader #2
PTP: high speed Paper Tape Punch
UP1: User defined Punch #1
UP2: User defined Punch #2
LPT: Line Printer
UL1: User defined List device
A: disk A
B: disk B
C: disk C
D: disk D for CP/M 2.0, up to 16 physical drives (P:)

Common File Types

.ASM standard assembler source files
.BAK backup file created by ED
.BAS BASIC file
.COM an executable memory image file
.DAT DATA file
.DOC messages or documentations
.FOR FORTRAN type file
.HEX an INTEL formatting HEX file (generated by ASM)
.INT an INTERMEDIATE file created by EBASIC and CBASIC
.LIB a LIBRARY file used by ED
.LST LiST file created by CBASIC
.MSG same as .DOC
.PRN printout of text formatters, compilers, assembler. . .
.REL relocatable files generated by MICROSOFT Basic-80
.SAV a system file on CP/M 2.0
.SUB source of SUBMIT utility
.SYM symbol table, generated by macro assemblers
.TEX ASCII data for Text-formatters
.TXT same as .DOC
.$$$ a TEMPORARY file

CP/M Version Comparison

CP/M 1.4 (change from 1.3)

Control/E and Control/R added as editing commands.

V and X commands added to editor.

Tabs echo as spaces in editor.

On ERA, system will ask ALL? after ERA *.*.

PIP has major upgrades in parameter selection and device names.

STAT displays size of each file.

Maximum of two disk drives.

Disk parameters in BDOS.

CP/M 2.0

Control/H backspaces correctly.

Control/J added for editing.

Control/R and Control/U improved.

Control/X backspaces and erases.

DIR improved to show entire directory on single screen.

ED defaults to V command and cannot access system files.

ERA erases files of current user only.

PIP has Gn, R, and W parameters added.

SAVE no longer modifies user memory area.

STAT has VAL:, DSK:, USR: and R added as well as fill attributes.

XSUB added.

USER added.

16 drives permitted to 16 megabytes.

More than 64 files may be in directory.

File access may be sequential or random.

Disk parameters are in BIOS.

BDOS functions 10, 15, 17, 18, 19, 22, 23, and 24 improved.

BDOS functions 18-36 added.

STAT

STAT cr	Calculates the storage remaining on all active drives.
STAT B: cr	Selects drive x, then calculates the storage.
STAT SAMPLE.* cr	Specifies a set of files to be scanned by STAT (here, all files with the primary name of SAMPLE).

STAT B: = R/O cr	Sets drive x to read only until a warm or cold boot takes place.
STAT DEV: cr	Produces a list of physical devices currently assigned to each logical unit.
STAT VAL: cr	Prints the possible values which can be assumed on for each logical device (CP/M 1.4).
STAT VAL: cr	Produces a summary of the new 2.0 commands.
STAT USR: cr	Produces a list of USER numbers which have files on the currently addressed disk.
STAT B:DSK:	Lists the drive characteristics of the disk B. B: can be in the range of A: –) P: (16 drives).
STAT SAMPLE.PRN $R/O cr	Sets the file SAMPLE.PRN to read only (CP/M 2.0).
STAT SAMPLE.PRN $R/W cr	Sets the file back to read/write (CP/M 2.0).
STAT B:PIP.COM $SYS cr	Makes PIP a system file–will not be displayed when DIR is issued (CP/M 2.0).
STAT B:PIP.COM $DIR cr	Makes PIP a system file–opposite of $SYS (2.0).

BDOS Function Summary

Function	Function Name	Input Parameters	Output Results
0	System Reset	none	none
1	Console Input	none	A = char
2	Console Output	E = char	none
3	Raw Console Input	none	A = char
4	Raw Console Output	E = char	none
5	List Output	E = char	none
*6	Direct Console I/O	see def	see def
7	**Not supported**		
8	**Not supported**		
9	Print String	DE = .Buffer	none
10	Read Console Buffer	DE = .Buffer	see def
11	Get Console Status	none	A = 00/01
12	Return Version Number	none	HL = Version #
13	Reset Disk System	none	see def
14	Select Disk	E = Disk Number	see def
15	Open File	DE = .FCB	A = Dir Code
16	Close File	DE = .FCB	A = Dir Code
17	Search for First	DE = .FCB	A = Dir Code
18	Search for Next	none	A = Dir Code
19	Delete File	DE = .FCB	A = Dir Code

20	Read Sequential	DE = .FCB	A = Err Code
21	Write Sequential	DE = .FCB	A = Err Code
22	Make File	DE = .FCB	A = Dir Code
23	Rename File	DE = .FCB	A = Dir Code
24	Return Login Vector	none	HL = Login Vect +
25	Return Current Disk	none	A = Cur Disk #
26	Set DMA Address	DE = .DMA	none
27	Get Addr (Alloc)	none	HL = .Alloc
*28	Write Protect Disk	none	see def
*29	Get R/O Vector	none	HL = R/O Vect +
*30	Set File Attributes	DE = .FCB	see def
*31	Get Addr (disk parms)	none	HL = .DPB
*32	Set/Get User Code	see def	see def
*33	Read Random	DE = .FCB	A = Err Code
*34	Write Random	DE = .FCB	A = Err Code
*35	Compute File Size	DE = .FCB	r0, r1, r2
*36	Set Random Record	DE = .FCB	r0, r1, r2
**37	Reset Drive	DE = drive vctr	A = Err Code
**38	Access Drive	DE = drive vctr	none
**39	Free Drive	DE = drive vctr	none
**40	Write Random zerofill	DE = .FCB	A = Err Code

*Available in CP/M 2.2 Version and later only, and MP/M.
**Available in MP/M only.
+ Note that A = L, and B = H upon return.

XDOS Function Summary

Function	Function Name	Input Parameters	Output Results
128	Absolute Memory Request	DE = .MD	A = Err Code
129	Relocatable Memory Request	DE = .MD	A = Err Code
130	Memory Free	DE = .MD	none
131	Poll	E = Device	none
132	Flag Wait	E = Flag	A = Err Code
133	Flag Set	E = Flag	A = Err Code
134	Make Queue	DE = .QCB	none
135	Open Queue	DE = .UQCB	A = Err Code
136	Delete Queue	DE = .QCB	A = Err Code
137	Read Queue	DE = .UQCB	none
138	Conditional Read Queue	DE = .UQCB	A = Err Code
139	Write Queue	DE = .UQCB	none
140	Conditional Write Queue	DE = .UQCB	A = Err Code
141	Delay	DE = #ticks	none
142	Dispatch	none	none
143	Terminate Process	E = Term. Code	none
144	Create Process	DE = .PD	none
145	Set Priority	E = Priority	none
146	Attach Console	none	none
147	Detach Console	none	none
148	Set Console	E = Console	none
149	Assign Console	DE = .APB	A = Err Code
150	Send CLI Command	DE = .CLICMD	none
151	Call Resident Sys Pr	DE = .CPB	HL = Result
152	Parse Filename	DE = .PFCB	see def
153	Get Console Number	none	A = Console #
154	System Data Address	none	HL = Sys Data Adr
155	Get Data and Time	DE = .TOD	none
156	Return Proc Dsc Adr	none	HL = proc descr adr
157	Abort Spec Process	DE = .ABTPB	A = Err Code

Appendix C

Hardware Suppliers

The listing of hardware suppliers in this appendix does not imply any endorsement as to quality or availability.

Adaptive Data and
Energy Systems
2627 Pomona Boulevard
Pomona, CA 91768
(714) 594-5858

Altos Computer Systems
2360 Bering Drive
San Jose, CA 95131
(408) 946-6700

CompuPro division, Godbout
Electronics
Box 2355
Oakland Airport, CA 94614
(415) 562-0636

Cromemco
280 Barnardo Avenue
Mountain View, CA 94043
(415) 964-7400

Delta Products
15392 Assembly Lane
Huntington Beach,
CA 92649
(714) 898-1492

Dynabyte
115 Independence Drive
Menlo Park, CA 94025
(415) 329-8021

Heath Company
Benton Harbor, MI 49022
(616) 982-3210

IBM Personal Computer
IBM, Dept. 765
Armonk, NY 10504

IMS International
Box 1
Carson City, NV 89701
(714) 978-6966

International Micro Systems
8425 Quivira Road
Lenexa, KA 66215
(913) 888-8330

Intertec Data Systems
2300 Broad River Road
Columbia, SC 29210
(803) 798-9100

Ithaca Intersystems
P.O. Box 91
Ithaca, NY 14850
(607) 257-0190

Lanier Business Products
1700 Chantilly Drive NE
Atlanta, GA 30324
(800) 241-1706

MicroDaSys, Inc.
2811 Wilshire Boulevard
Santa Monica, CA 90403
(213) 829-6781

Micropolis
21329 Nordhoff Street
Chatsworth, CA 91311
(213) 709-3300

Morrow Designs
5221 Central Avenue
Richmond, CA 94804
(415) 524-2101

NEC
1401 Estes Avenue
Elk Grove Village, IL 60007

NNC Electronics
15631 Computer Lane
Huntington Beach,
CA 92609
(714) 895-8000

Quasar Data Products
10330 Brecksville Road
Brecksville, OH 44141
(216) 526-0838

Quay Corporation
P.O. Box 386
Freehold, NJ 07728
(201) 681-8700

SD Systems
P.O. Box 28810
Dallas, TX 75228

Systems Group
867 North Main Street
Orange, CA 92668
(714) 633-4460

TEI
5075 South Loop East
Houston, TX 77033
(713) 738-2300

Tarbell Electronics
Suite B
950 Dovlen Place
Carson, CA 90746
(213) 538-4251

Televideo Systems
1170 Morse Avenue
Sunnyvale, CA 94086
(408) 745-7760

Vector Graphics
313 Via Colinas
Westlake Village, CA 91362
(213) 991-2303

Wang
One Industrial Avenue
Lowell, MA 01851
(800) 225-0643

Zenith Data Systems
Hilltop Road
St. Joseph, MI 49085

Zobex
P.O. Box 1847
San Diego, CA 92112
(714) 296-9182

Appendix D

Software Suppliers

The listing of software suppliers in this appendix does not imply any endorsement as to quality or availability.

A list of software vendors is given first, each with an assigned vendor number. Then the CP/M product listing is given, with vendors referenced by vendor number.

Note: Many of the names of software products are trademarked by the company producing the software.

A-T Enterprises (1)
221 North Louis Street
La Habra, CA 90631

AArdvark Software (2)
783 North Water Street
Milwaukee, WI 53202
(414) 289-9988

AMSI (3)
Suite 200
1935 Cliff Valley Way NE
Atlanta, GA 30329
(404) 634-9535

Aaron Associates (4)
Box 170A
Garden Grove, CA 92640

Accountants Microsystems, Inc. (5)
1404 140th Place NE
La Habra, CA 90631

Advanced Micro Techniques (6)
1291 East Hillsdale Boulevard #209
Foster City, Ca 94404
(415) 349-9336

All Systems (7)
332 East 30th Street
New York, NY 10016
(800) 221-2486

Allen Ashley (8)
395 Sierra Madra Villa
Pasadena, CA 91107
(213) 793-5748

Anderson-Bell Company (9)
2916 South Stuart Street
Denver, CO 80236

Arkansas Systems (10)
Suite 206
8901 Kanis Road
Little Rock, AR 72205
(501) 227-8471

Arkenstone (11)
Suite 369
870 Market Street
San Francisco, CA 94102

Asyst Design Services (12)
756 Bowling Green
Corland, NY 13045
(607) 756-8247

Avocet Systems (13)
804 South State Street
Dover, DE 19901
(302) 734-0151

Brook Chambery
Software Division (14)
900 Culver Road
Rochester, NY 14609

Business Micro Products (15)
609 South Livermore Avenue
Livermore, CA 94550
(415) 449-4412

COMM SOFT (16)
665 Maybell Avenue
Palo Alto, CA 94306
(415) 493-2184

CPAids (17)
1640 Franklin Avenue
Kent, OH 44240
(800) 321-2430

Cal Data Systems (18)
Suite B
150 Felker Street
Santa Cruz, CA 95060

Cexec (19)
Suite 300
211 Sutter Street
San Francisco, CA 94108
(415) 981-4724

Chameleon Software, Inc. (20)
4733 North Mitchner
Indianapolis, IN 48226
(317) 545-5098

Chang Laboratories (21)
10228 North Stelling Road
Cupertino, CA 95014
(408) 725-8088

Compiler Systems, Inc. (22)
P.O. Box 145
Sierra Madre, CA 91024
(213) 355-1063

CompuServe (23)
5000 Arlington Centre Boulevard
Columbus, OH 43220
(614) 457-8600

Compumax (24)
P.O. Box 1139
Palo Alto, CA

Computer Pathways Unlimited (25)
2151 Davcor Street SE
Salem, OR 97302
(503) 363-8929

Computer Products, Inc. (26)
P.O. Box 805
Mesa, AZ 85202

Computer Services (27)
P.O. Box 2292
Hickory, NC 28601
(704) 294-1616

Compuview Products (28)
618 Louise
Ann Arbor, MI 48103
(313) 996-1299

Concomp Industries (29)
8338 Center Drive
La Mesa, CA 92041
(714) 464-6373

Condor Computer Corporation (30)
P.O. Box 8318
Ann Arbor, MI 48107
(313) 769-3988

Creative Computing Software (31)
Morris Plains, NJ 07950
(800) 631-8112

Custom Electronics, Inc. (32)
238 Exchange Street
Chicopee, MA 01013
(413) 592-4761

Cybernetics (33)
Suite 208
8041 Newman Avenue
Hungtington Beach, CA 92647
(714) 848-1922

Data Train (34)
Suite 3
840 NW 6th
Grants Pass, OR 97526
(503) 476-1467

DataSort, Inc. (35)
Suite 202
3237 Claremont Way
Napa, CA 94558

Datastat Systems (36)
631 B Street
San Diego, CA 92101

Dataword, Inc. (37)
1404 140th Place NE
Bellevue, WA 98007
(206) 643-2050

Delphic Systems (38)
2260 Compton Avenue
St. Louis, MO 63104

Designer Software (39)
Suite 718
3400 Montrose Boulevard
Houston, TX 77006
(713) 520-8221

Digiac Corporation (40)
175 Engineers Road
Smithtown, NY 11787
(516) 273-8600

Digital Contructs (41)
130 Easst Main Street
Norristown, PA 19401
(215) 279-5652

Digital Electronics Systems (42)
Box 5252
Torrance, CA 90510
(213) 539-6239

Digital Marketing (43)
2670 Cherry Lane
Walnut Creek, CA 94596
(415) 938-2880

Digital Research (44)
801 Lighthouse Avenue
Pacific Grove, CA 93950

Digital Systems (45)
4301 Montana
El Paso, TX 79903
(915) 566-8655

dilithium Press (46)
Suite E
11000 SW 11th Street
Beaverton, OR 97005
(503) 646-2713

Douthett Enterprises (47)
906 North Main
Wichita, KA 67203

Dynamic Microprocessor Associate (48)
545 Fifth Avenue
New York, NY 10017
(212) 687-7115

EMERTUS (49)
1932 South Karen Avenue
Freso, CA 93727
(209) 251-3525

EZ Software (50)
P.O. Box 591
Novato, CA 94947

Ecosoft (51)
P.O. Box 68602
Indianapolis, IN 46268
(317) 283-8883

Efficient Management Systems (52)
Suite 304
3645 Grand Avenue
Oakland, CA 94610

Ellis Computing (53)
600 41st Avenue
San Francisco, CA 94121
(415) 751-1522

End User Software (54)
150 Felker Street
Santa Cruz, CA 95060

FMG Corporation (55)
Suite 13
5280 Trail Lake Drive
Fort Worth, TX 76133
(817) 294-2510

Fair Com (56)
2606 Johnson Drive
Columbia, MO 65201
(314) 445-3304

Ficomp (57)
3017 Talking Rock Drive
Farfax, VA 22031

Fox and Geller Associates (58)
P.O. Box 1053
Teaneck, NJ 07666
(201) 837-0142

Friends Software (59)
P.O. Box 527
Berkeley, CA 94701
(415) 540-7282

Graham-Dorian Software Systems (60)
P.O. Box 16355
Forth Worth, TX 76133
(817) 294-2510

HSC Computer Services (61)
P.O. Box 43
Brooklyn, NY 11236

Information Engineering (62)
Box 305
New Market, NH 03857
(603) 659-5891

Infosoft (65)
25 Sylvan Road South
Westport, CT 06880
(203) 226-8937

InnoSys, Inc. (66)
2150 Shattuck Avenue
Berkeley, CA 94704
(415) 843-8122

Innovative Software (68)
8905 Holly
Kansas City, MO 64114
(816) 942-8778

Innovative Software Applications (69)
Box 2797
Menlo Park, CA 94205
(415) 326-0805

Interactive Microware (70)
Box 771
Satate College, PA 16801
(814) 238-8294

International Micro Systems (71)
3077 Merriam Lane
Kansas City, KA 66106

Intersystems (72)
P.O. Box 91
Ithica, NY 14850
(607) 257-0190

JRT Systems (73)
P.O. Box 22365
San Francisco, CA 94122
(415) 566-4240

Johnson-Laird, Inc. (74)
1556 SW 66th
Portland, OR 97225
(503) 292-6330

Kit Bits, Inc. (75)
Box 592293
Miami, FL 33159

Laboratory Microsystems (76)
4147 Beethoven Street
Los Angeles, CA 90066
(213) 390-9292

Lexisoft (77)
Box 267
Davis, CA 95616
(916) 758-3630

Lifeboat Associates (78)
2248 Broadway
New York City, NY 10024

Londe Parker Michels, Inc. (79)
150 North Meramac
St. Louis, MO 83105
(314) 725-5501

MICAH (80)
Box 4987
Walnut Creek, CA 94596
(415) 933-2783

MT Microsystems (81)
1562 Kings Cross Drive
Cardiff, CA 92007
(714) 753-4856

Mark of the Unicorn (82)
P.O. Box 423
Arlington, MA 02174
(617) 489-1387

Metasoft Corporation (83)
Suite E
711 East Cottonwood Lane
Casa Grande, AZ 85222
(602) 836-6160

Micro Applications Group (84)
7300 Caldus Avenue
Van Nuys, CA 90146

Micro Architech Inc. (85)
96 Dothan Street
Arlington, MA 02174
(617) 643-4713

Micro Business Systems (86)
6401 Roosevelt Way NE
Seattle, WA 98115

Micro Byte Computer Store (87)
2626 Union Avenue
San Jose, CA 95124

Micro Computer Management, Inc. (88)
P.O. Box 1794
Ft. Collins, CO 85022
(303) 493-5700

Micro Data Base Systems, Inc. (89)
Box 248
Lafayette, IN 47902
(317) 448-1616

Micro Focus, Inc. (90)
1601 Civic Center Drive
Santa Clara, CA 95050
(408) 984-6961

Micro Source (91)
1425 West 12th Place
Tempe, AZ 85281

Micro-Ap (92)
9807 Davona Drive
San Ramon, CA 94583
(415) 828-6697

Micro-Call Services (93)
9655-M Homestead Court
Laurel, MD 20810

MicroTech Exports, Inc. (94)
Suite 2
467 Hamilton Avenue
Palo Alto, CA 94301
(415) 324-9114

Microcomputer Consultants (95)
P.O. Box T
Davis, CA 95617
(916) 756-8104

Microed (96)
391 O Bandini Street
San Diego, CA 92103
(714) 299-1125

Micropro (97)
1299 4th Street
San Rafael, CA 94901

Microsoft (98)
Suite 200
400 108th NE
Bellevue, WA 98004
(206) 455-8080

Microsoftware Producers (99)
3109 Filmore Street
San Francisco, CA 94123
(415) 346-7025

Microstuf, Inc. (100)
Box 33337
Decatur, GA 30033
(404) 491-3787

Microtax (101)
Suite F
22713 Ventura Boulevard
Woodland Hills, CA 91364
(213) 704-7800

Microtex (102)
3600 Wilshire Boulevard #1510
Los Angeles, CA 90010
(213) 738-9972

Midwest Micro-Tek (103)
P.O. Box 29411
Brooklyn Center, MN 55429

Mitchell E. Timin Engineering (104)
Suite E2
9575 Genesee Avenue
San Diego, CA 92121
(714) 455-9008

Monoson Microsystems (106)
51 Main Street
Watertown, MA 02172
(617) 924-2124

Mycroft Labs (107)
P.O. Box 6045
Tallahasee, FL 32301

Nautilus Systems (108)
P.O. Box 1098
Santa Cruz, CA 95061

North America MICA, Inc. (109)
11772 Sorrento Valley Road #240
San Diego, CA 92121

Northwest Analytical (110)
P.O. Box 14430
Portland, OR 97214
(503) 238-9760

Oasis Systems (111)
2765 Reynard Way
San Diego, CA 92103
(714) 291-9489

Omikron (112)
1127 Hearst Street
Berkeley, CA 94702
(415) 845-8013

Ontrak (113)
P.O. Box 2134
Yorba Linda, CA 92686
(714) 993-0142

Oregon Professional Microsystems
(114)
4110 NE Alameda
Portland, OR 97212
(503) 282-5835

Organic Software (115)
1492 Windsor Way
Livermore, CA 94550
(415) 455-4034

Orthocode Corporation (116)
P.O. Box 6191
Albany, CA 94706
(415) 527-9300

Osborne Associates (117)
Box 2036
Berkeley, CA 94702

PCD Systems (118)
P.O. Box 143
Penn Yan, NY 14527
(315) 536-3734

Peachtree Software (119)
Suite 700
3 Corporate Square
Atlanta, GA 30329
(404) 325-8533

Pickles & Trout (120)
P.O. Box 1206
Goleta, CA 93017
(805) 967-9563

Provar, Inc. (121)
Box 2280
Hammond, IN 46323
(312) 374-7335

R & B Computer Systems (122)
1954 East University
Tempe, AZ 85281
(800) 528-7385

Rothberg Information Systems
(123)
260 Sheridan Avenue
Palo Alto, CA 94306

SP80 (124)
P.O. Box 2745
Reston, VA 22091
(703) 620-4639

Select Information Systems (125)
919 Sir Francis Drake Boulevard
Kentfield, CA 94904
(415) 459-4003

Small Business Applications (126)
Suite 205
3220 Louisiana
Houston, TX 77006
(713) 528-5158

SoHo Group (127)
Suite 4-B
140 Thompson Street
New York, NY 10012

Software Development & Training (128)
P.O. Box 4511
Huntsville, AL 35802

Software Establishment (129)
705A Lakefield Road
Westlake Village, CA 91361
(213) 991-7711

Software Hows (130)
P.O. Box 36275
Los Angeles, CA 90036
(213) 731-0876

Software Products International (131)
Suite 115
5482 Complex Street
San Diego, CA 92123
(714) 268-4346

Software Review (132)
704 Solano Avenue
Albany, CA 94704
(415) 527-7730

Software Store (133)
706 Chippewa Square
Marquette, MI 49855
(906) 228-7622

Sorcim (134)
405 Aldo Avenue
Santa Clara, CA 95051
(408) 727-7634

Southern Computer Systems (135)
P.O. Box 3373A
Birmingham, AL 35255
(205) 933-1549

Southern Digital Systems (136)
Suite 806A
Vernon Park Mall
Kingston, NC 28501
(919) 522-2327

Standard Software Corporation (137)
10 Mazzeo Drive
Randolph, MA 02368
(617) 963-7220

Statcom Corporation (138)
Suite 202
5766 Balcones
Austin, TX 78731
(512) 451-0221

Stok Computer Interface (139)
P.O. Box 501
Woodside, NY 11377
(212) 426-7022

Structured Systems (140)
5204 Claremont
Oakland, CA 94618
(415) 547-1567

Supersoft Associates (141)
P.O. Box 1628
Champaign, IL 61820
(217) 344-7596

Systemation (142)
P.O. Box 75
Richton Park, IL 60471
(312) 481-2420

Systems Plus (143)
3975 East Bayshore
Palo Alto, CA 94303
(415) 969-7047

Systems Shoppe (144)
104 East Main
Greenwood, MO 64034
(816) 537-7927

Tarbell Electronics (145)
Suite B
950 Dovlen Place
Carson, CA 90746
(213) 538-4251

Taxmasters (146)
P.O. Box 5030
San Clemente, CA 92672
(714) 492-7633

The Code Works (147)
Box 550
Goleta, CA 93116
(805) 683-1585

The Software Store (148)
706 Chippewa Square
Marquette, MN 49855
(906) 228-7622

Time Management Software (149)
P.O. Box 727
Cushing, OK 74023
(800) 824-7888

Timin Engineering Company (150)
9575 Genesee Avenue
San Diego, CA 92121
(714) 455-9008

Tiny C Associates (151)
P.O. Box 269
Holmdel, NJ 07733
(201) 671-2296

Uevon Computer Systems, Inc. (152)
899 Logan Street
Denver, CO 80203
(800) 525-1637

Universal Software Studios (153)
179 West Smith Street
Gallatin, TN 37066
(615) 452-1027

Urban Software (154)
19 West 34th Street
New York, NY 10001
(212) 947-3811

Vandata (155)
17541 Stone Avenue North
Seattle, WA 98133
(206) 542-8370

Vanloves/Vital Information, Inc. (156)
7899 Mastin Drive
Overland Park, KS 66204
(800) 255-5119

Walt Bilofsky (157)
14478 Glorietta Drive
Sherman Oaks, CA 91423
(213) 986-4885

Westico (158)
25 Van Zant Street
Norwalk, CT 06855
(203) 853-6880

Whitesmiths (159)
P.O. Box 1132
Ansonia Station
New York, NY 10023
(212) 799-1200

Winterhalter & Associates (160)
3825 North Zeeb Road
Dexter, MI 48130
(313) 665-5582

CP/M Product Listing

Class	Application	Product	Vendor
Business	Account Recv.	(Business)	35
Business	Accounting	(Business)	33
		(Business)	113
		(Business)	4
		(Business)	10
		(Business)	17
		(Business)	140
		(Business)	119
		(Business)	54
		(Business)	130
		(Business)	86
		(Business)	60
		Accounting Plus	143
		CBS	78
		Datawrite	37
		General On-Line Database	47
		Osborne Business	45
		TCS Business	7
		The Bookkeeper System	153
		The Boss	78
Business	Calendar	Ticker File/Appt. Calendar	11
Business	Construction	(Business)	60
Business	Date Entry	Datastar	97
Business	Dental	(Business)	60
		Dental Billing	7
		Microdent	130
Business	Energy	Bill	79
		Lifecost	79
		Overhang	79
Business	Financial Model	T/Maker II	78
Business	Financial Model	Forecaster	129
		FPL	78
		Logicalc	131
		Microplan	137, 21
		PascalC	29
		Plan80	43
		Supercalc	134
		T/Maker	78
Business	Genealogy	Roots/M	16
Business	Income Property	INCOPROP	50
Business	Inventory	(Inventory Control)	95
		(Inventory)	46
		Inventory Control	158
Business	Job Cost	Job Cost Control	158
Business	Labor-Cost	(Business)	18
		(Business)	18
Business	Legal	ESQ-1	78
		Legal Billing	7
Business	Loans	Financier	5
Business	Mailing List	Executive Mailout	46
		Mailing List	11
		Mailing List	43

Class	Application	Product	Vendor
		NAD	78
		Postmaster	78
Business	Medical	(Business)	60
		Medical Billing	7
		Micromed	130
Business	Pert	Milestone	43
		Project Management System	109
Business	Project Manage	Task	3
Business	Property	Emertus Property Management	49
Business	Scheduling	Databook II	43
Business	Solar Design	Pascode 1	79
Business	Sort	QSORT	78
		SORT/B	142
		Ultrasort	43
		VSORT	78
Business	Sorting	Supersort I	97
Business	Tax	Microtax	101
		Personal Tax	2
		Standard Tax	78
Business	Time Management	Angel	149
		Guardian	149
Business	Whatif	Worksheet	127
Communications		(Communications)	36
		AMCALL	43
		BSTAM	78
		BSTMS	78
		CLINK	107
		CP/Modem	64
		Crosstalk	100
		MCALL	43
		Plink	78
		Telecom	145
		Term	141
Communications	Control	Remote	100
Development	ADA	ADA	42
Development	ALGOL	ALGOL-60	78
Development	APL	APL/V80	78
Development	Assembler	(Assembler)	26
		(Assembler)	98
		(Assembler)	103
		I/SAL	65
		MAC	44
		MAC	44
		MACRO-80	98
		PASM	78
		SMAL/80	78
Development	Basic	Basic Compiler	98
		Basic-80	98
		Bazic	43
		CB80	22
		CBASIC2	43, 44, 78
		CBASIC2	44

Class	Application	Product	Vendor
		Northstar Basic	65
		NSBASIC Interface	65
		Tarbell Basic	145
		XYBasic	78
Development	C	BDS C Compiler	78
		C	141
		C	159
		C Compiler	147
		C/80	2
		Tiny C One	151
Development	C	Tiny C Two	151
		Whitesmiths C Compiler	78
Development	Cobol	CIS Cobol	90
		Cobol-80	98
		Nevada Cobol	53
		NPS-Micro Cobol	132
Development	Crossassembler	Cross Assemblers	13
		XASM	78
		Z8000 Crossassembler	141
Development	Crosscompiler	Cross-Compile Forth	108
		Trans 86	43
Development	DBMS	Basic B +	38
		BT-80	44
		Condor Series 20/rDBMS	30
		FMS-80	143
		IDM-C1	85
		KBASIC	78
		Magsam III	84
		MBASIC IV	84
		MDBS III	89
		MICRO B +	56
		MIDAS	123
		Optimum	152
		Prism/LMS	84
		Quest	131
		Selector IV	92
		Tarbell DBMS	145
		Whatisit	78
Development	Debugger	Bug	78
		DEB/ZEB	65
		Raid	78
		SID	44
Development	Disassembler	DISSAX	19
		V-Com Disassembler	28
Development	Disk Reformatter	Reformatter	94
Development	Forth	Fig Forth	105
		Forth	141
		Z-80 Forth	76
Development	FORTRAN	FORTRAN IV	141
		FORTRAN-80	98
Development	Indexing	FABS	43
Development	Info Management	Leverage	154

Class	Application	Product	Vendor
Development	ISAM	KSAM80	52
Development	Language	PL/I-80	44
Development	Linker	Linka	65
Development	Lisp	Mulisp	98
Development	List Manager	Analyst	140
Development	Macro Assembler	Act I	43
		Act II	43
Development	OS	Stok Pilot	139
Development	Pascal	Pascal	159
		Pascal/M	43
		Pascal/MT +	81
		Pascal/Z	72
		Speed Programming Package	81
		Tiny Pascal	141
Development	Program Writing	Genesis	149
		Pearl–Level 2	43
		Pearl–Level 3	43
		Pearl–Level 1	43
		The Program Writer/Reporter	156
Development	RATFOR	RATFOR	141
		RATFOR	2
Development	Report Gen	The Formula	48
Development	RPG	ACCESS/80	59
Development	Screen Editor	Quickscreen	58
Development	Utilities	Interchange	51
		Utilities	141
Development	Disassembler	DISTEL	78
Games		(Games)	31
Games	Adventure	Adventure	31
Games	D & D	Dungeon Master	141
		Nemesis	141
Games	Eliza	Analiza	141
Maintenance	Disk	Disk Doctor	141
Maintenance		Diagnostics II	141
Maintenance	Disk	DPATCH	6
OS		CP/M-86	44
		CP/NET	44
Personal	Banking	Money Maestro	67
Research		MUSIMP	98
Research	Statistics	ABSTAT	9
		Microstat	51
Utility		(Utilities)	57
		(Utilities)	80
		Basic Utility Disk	78
		IBM/CPM	78
		PBLC-DMN-1	145
		PBLC-DMN-2	145
		Program Map	133
		Reclaim	78
		String/80	78
		The String Bit	78
		Unprotect	142

Class	Application	Product	Vendor
Utility	Disk	Dif and Del	41
		Disk Fix	133
		Diskcopy	11
Utility	Spooler	Despool	44
		KLH Spooler	145
Word Processing	Data Entry	RADAR	135
Word Processing	Dictionary	Diction II	43
Word Processing	Editor	CRTFORM	138
		Edit	78
		EDIT-80	98
		MINCE	78, 82
		PMATE	78
		Scribble	82
		Star-Edit	141
		TFS	141
		VEDIT	28
		Wordmaster	97
Word Processing	Form Handling	FRMFLEX	19
Word Processing	Indexing	Documate/Plus	116
Word Processing	Letters	Letterright	140
Word Processing	Mailing	Mailmerge	65
		Super M List	141
Word Processing	Screen Editor	Forms 2	78
Word Processing	Spelling	Copyproof	43
		Microproof	149
		Microspell	78
		Spellbinder	77
		Spellguard	69
		Spellstar	97
		The Word	111
Word Processing	Word Processing	Amethyst	82
		Copywriter	43
		Magic Menu	43
		Magic Wand	126
		Select	114, 125
		TEX	44
		Textwriter	115
		Texwriter III	78
		The Benchmark	83, 122
		Txtbook	19
		Word	85
		Wordstar	97
		WPDAISY	65

CP/M Suppliers for TRS-80 Systems

FMG Corporation
P.O. Box 16020
Fort Worth, TX 76133
(817) 738-0251

Lifeboat Associates
1651 Third Avenue
New York, NY 10021
(212) 860-0300

Cybernetics
Suite 208
8041 Newman Avenue
Huntingon Beach, CA 92647

Pickles & Trout
P.O. Box 1206
Goleta, CA 93116
(805) 685-4641

Appendix E

CP/M Diagnostics

Problem	Cause	Solution
CP/M does not come up. With disk drive cover removed, head does not load to disk.	Disk inserted incorrectly.	Insert disk correctly.
	Hardware problems.	Check disk cables, drive, and disk controller.
CP/M does not come up. With disk drive cover removed, head loads but does not move across disk.	Problems with SBOOT program, wrong memory size or bad CPU or memory (first page).	Check lowest page of memory, SBOOT, CPU.
CP/M does not come up. With disk drive cover removed head loads and moves very slightly across disk. No display on video.	Wrong port numbers or jumps in CONOT of BIOS, bad upper memory.	Stop computer and examine memory where system is stopped. If in CONOT, change port or jump instructions. Check upper memory.
CP/M does not come up. Some video displayed, but text is garbled.	Problems in video terminal, baud rates of terminal or I/O card, or jump instructions in CONOT of BIOS.	Check video terminal switches and CONOT of BIOS.
CRC error messages or write error messages.	Bad disks, dirty disk head or bad head alignment.	Try different disk, do disk tests, clean head with alcohol. If this doesn't work, get disk alignment checked.
Read error messages.	Bad disks, dirty disk head or bad head alignment.	Do disk test, clean heads. If these do not work, check alignment.
Disk does rhythmic head load and unload continuously. Control/C does not stop cycle.	Disk has run out of storage space, and BIOS or program does not recover properly.	Reboot system. Check storage space. Remember editor and PIP require .$$$ files with as much space as original file. Clear off unused files as BAK files and try again.
System locks up and freezes on a call to printer.	Printer is not turned on.	Turn printer on.

Problem	Cause	Solution
System locks up and freezes. No printer call was issued.	Hardware problems in CPU or memory.	Check memory and CPU.
	Printer not turned on.	Turn Printer on.
Assembler generates error messages even though source file is correct. Error messages indicate source file is not correct.	Poorly designed memory cards are overloading the bus or memory is bad.	Minimize memory size to only what is necessary for compile. Check memory.
Computer hung in disk or print cycle and Control/C doesn't seem to stop anything.		Stop system with Control/S, then use Control/C.
R/O error message.	Disks were changed without a warm start.	Whenever disks are changed, a warm start (Control/C) must be done, or a RESET before change if BASIC is being used. Warm start and try again.

Appendix F

CP/M User Groups

One of the most remarkable sources of CP/M programs is the CP/M User's Groups. Operated out of the Lifeboat Associates address, this group has over 60 different disk volumes available at $8 each after a $4 registration charge.

There is no restriction on copying these disks, and often computer clubs in cities order a set and local club members copy programs as they are needed. Another alternative is to locate a few local CP/M users, order most of the set, and split the cost. Each person then copies those programs he needs and then the disks are stored at a bank deposit box.

The disks include utility programs, games, documentation on features of CP/M not covered in the distributor's manuals, and application programs (Table E.1). Some of the software is high quality, while some may have little value to you. You will have fun searching these disks for treasures for your own use.

You can also help the User's Group by contributing your own software and creativity. If the programs help you, help others by sending programs that you develop to:

CP/M User's Group
2248 Broadway
New York, NY 10024

CP/M User Group

Volume	Description
1	CP/M Utilities
2	Lawrence Livermore BASIC
3	BASIC E Games
4	ACTOR Interpreter and Fortran Random Number Subroutines
5	Mixed Games
6	Chicago Area Computer Hobbyist Exchange Programs (Mailing List Program)
7	PILOT
8	Utilities
9	General Ledger Program
10	Lawrence Livermore BASIC (Updated)
11	Disk Tiny BASIC and PT BASIC/5
12	PILOT (Update)

13	Mixed Games
14	CP/M Utilities
15	Utilities and Non-BASIC Games
16	Utilities, FOCAL
17	Utilities, Denvery Tiny BASIC, Games
18	Math Routines
19	Utilities
20	BASIC E Games
21	Microsoft BASIC Games
22	Monster Startrek Games
23	STOIC Compiler
24	CP/M utilities, macrolibraries, RATFOR
25	Utilities, modem software, disk catalog system
26	Microsoft BASIC Games
27	Microsoft BASIC Games
28	Tarbell Data Base System and Algol-like Language
29	Assembler Games, Some BASIC E Source
30	BASIC E PLM Source
31	Tarbell BASIC and Source (Part I)
32	Tarbell BASIC Source (Part II)
33	Search and Rescue Programs
34	SAM 76 Language
35	FELIX Graphics Animation System
36	Assemblers and Editors
37	CBASIC2 Programs
38	CP/M Speed-up Utilities and Tarbell Controller Utilities
39	Music Programs
40	Utilities (Modem, DU, Cataloging)
41	Ham Radio Programs
42	Disassemblers
43	Osborne A/P
44	Osborne G/L
45	Osborne Payroll
46	Modem Programs and Utilities
47	Modem Programs
48	BDOS C Programs
49	CDOS Disk and RATFOR
50	Pascal Compiler
51	Stage 2 Microprocessor

52	Copyfast and Other Utilities
53	BDOS-C Adventure
54	MBASIC Games
55	Original Adventure (run time)
56	Original Adventure (source)
57	Expanded Adventure (run time)
58	CP/M Utilities
59	Diagnostics
60	6502 Simulator
61	Bulletin Board Software
62	PASCAL and Communication Software
63	Utilities
64	Games, Disassembler

If you bought this book looking for a CP/M Adventure game, you have just found it (Volume 11)! The game is available from the SIG/M user Group along with other disk volumes. Each disk is in the public domain. We suggest ordering disks through your local CP/M club or getting several users together and splitting the cost. Each of these disks is $6 each for U.S. mailing. For a SASE, you can get a catalog listing of all volumes. Order from SIG/M, Box 97, Iselin, NJ 08830. (The Adventure disk is Volume 11). The same disks are also available through the New York CP/M Users Group and are numbered as Volumes 55-57.)

SIG/M User Group

Volume	Description
1	Original Adventure Run-time Code
2	Original Adventure–Source Code
3	550 Point Expanded Adventure (run-time code–Z-80 only)
4	CP/M Utilities
5	ICOM Disk Controller Diagnostics
6	6502 Simulator
7	Bulletin Board Software
8	Pascal Compiler and Modem Programs
9	Utilities
10	Games and Utilities (Heath Terminal)

11	550 Point Expanded Adventure (8080 or Z80)
12	Catalog of New York User Group Programs and SIG/M Programs
13	FIG-FORTH
14	HELP File System
15	New York User Group Catalog
16	CP/M Utilities (DU, etc.)
17	CP/M Utilities
18	CP/M Utilities
19	Pascal/Z Programs
20	PCE System Monitor
21	Pascal/Z Utilities
22	Pascal/Z Utilities
23	MBASIC Disassembler and Utilities
24	Pascal/Z Utilities
25	Pascal/Z Database Seed System

Other User Groups

Amethyst User Group
Barry Dobyns
1633 Royal Creat #1128
Austin, TX 78741

BDOS-C User Group
Box 287
Yates Center, KS 66783

Osborne Business Software User Group
Suite 11
2256 Main Street
Otay, CA 92011

Pascal/Z User Group
7962 Center Parkway
Sacramento, CA 95823

PL/1 User Group
Digital Research
Box 579
Pacific Grove, CA

Appendix G

Operational Procedures

1. Always back up master program disks upon purchase, then use the back-up disk and store the master disk at a separate georgraphic location.
2. Back up data and program disks periodically.
3. Keep disks in their envelopes unless in use. Never expose them to extremes of heat and cold. Handle them only by their plastic covers. Write on labels (not disks) and then apply the labels.
4. Include a date and title within the program itself as you develop it.
5. Organize programs on disks by *use*—i.e., all Microsoft games together on a disk, all CBASIC games on a disk, all general ledger programs on a disk, etc.
6. Use the same memory size CP/M on all disks. If you update the CP/M BIOS, keep track of disks with the new CP/M.
7. Keep at least one back-up copy of everything.

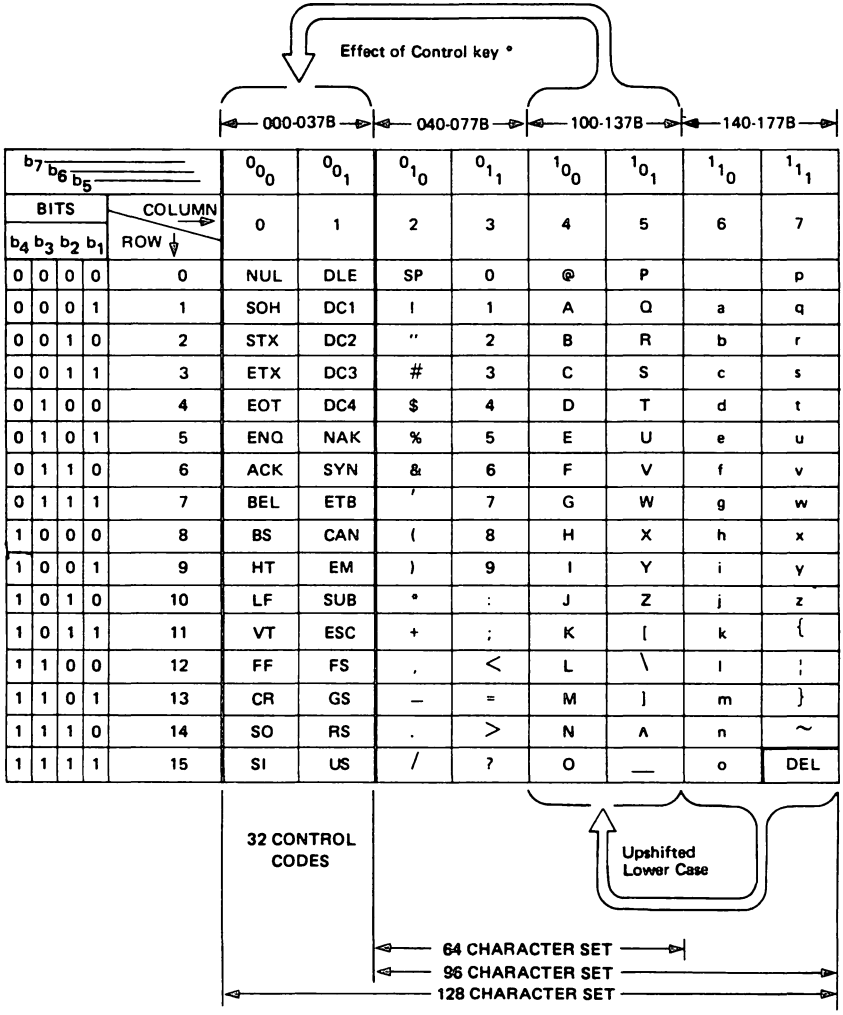
Appendix H

ASCII Codes

ASCII Code	Character	ASCII Code	Character
000	NUL	032	SPACE
001	SOH	033	!
002	STX	034	"
003	ETX	035	#
004	EOT	036	$
005	ENQ	037	%
006	ACK	038	&
007	BEL	039	'
008	BS	040	(
009	HT	041	)
010	LF	042	*
011	VT	043	+
012	FF	044	,
013	CR	045	-
014	SO	046	.
015	SI	047	/
016	DLE	048	0
017	DC1	049	1
018	DC2	050	2
019	DC3	051	3
020	DC4	052	4
021	NAK	053	5
022	SYN	054	6
023	ETB	055	7
024	CAN	056	8
025	EM	057	9
026	SUB	058	:
027	ESCAPE	059	;
028	FS	060	<
029	GS	061	=
030	RS	062	>
031	US	063	?
064	@	096	`
065	A	097	a
066	B	098	b
067	C	099	c
068	D	100	d
069	E	101	e
070	F	102	f

ASCII Code	Character	ASCII Code	Character
071	G	103	g
072	H	104	h
073	I	105	i
074	J	106	j
075	K	107	k
076	L	108	l
077	M	109	m
078	N	110	n
079	O	111	o
080	P	112	p
081	Q	113	q
082	R	114	r
083	S	115	s
084	T	116	t
085	U	117	u
086	V	118	v
087	W	119	w
088	X	120	x
089	Y	121	y
090	Z	122	z
091	[	123	{
092	\	124	|
093	]	125	}
094	^	126	~
095	_	127	DEL

ASCII codes are in decimal
LF = Line Feed, FF = Form Feed, CR = Carriage Return,
DEL = Rubout



b_7 b_6 b_5 →					0_0	0_0	0_1	0_1	1_0	1_0	1_1	1_1
BITS				COLUMN →	0	1	2	3	4	5	6	7
b_4	b_3	b_2	b_1	ROW ↓								
0	0	0	0	0	NUL	DLE	SP	0	@	P		p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	–	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

EXAMPLE: The representation for the character "K" (column 4, row 11) is.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1
BINARY	1	0	0	1	0	1	1
OCTAL	1		1			3	

* Depressing the Control key while typing an upper case letter produces the corresponding control code on most terminals. For example, Control-H is a backspace.

8080 Codes

Mnemonics		oct	hex	States	Description
ADD	A	207	87	4	Double A (shift A left)
ADD	B	200	80	4	Add B to A
ADD	C	201	81	4	Add C to A
ADD	D	202	82	4	Add D to A
ADD	E	203	83	4	Add E to A
ADD	H	204	84	4	Add H to A
ADD	L	205	85	4	Add L to A
ADD	M	206	86	7	Add LOC(H&L) to A
ADI	v	306	C6	7	Add v to A
ADC	A	217	8F	4	Double A with carry (shift left with carry)
ADC	B	210	88	4	Add B to A with carry
ADC	C	211	89	4	Add C to A with carry
ADC	D	212	8A	4	Add D to A with carry
ADC	E	213	8B	4	Add E to A with carry
ADC	H	214	8C	4	Add H to A with carry
ADC	L	215	8D	4	Add L to A with carry
ADC	M	216	8E	7	Add LOC(H&L) to A with carry
ACI	v	316	CE	7	Add v to A with carry
ANA	A	247	A7	4	Test A and clear carry
ANA	B	240	A0	4	AND B with A
ANA	C	241	A1	4	AND C with A
ANA	D	242	A2	4	AND D with A
ANA	E	243	A3	4	AND E with A
ANA	H	244	A4	4	AND H with A
ANA	L	245	A5	4	AND L with A
ANA	M	246	A6	7	AND LOC(H&L) with A
ANI	v	346	E6	7	AND v with A
CALL	aa	315	CD	17	Call subroutine aa
CZ	aa	314	CC	11/17	If zero CALL
CNZ	aa	304	C4	11/17	If not zero CALL
CP	aa	364	F4	11/17	If plus CALL
CM	aa	374	FC	11/17	If minus CALL
CC	aa	334	DC	11/17	If carry CALL
CNC	aa	324	D4	11/17	If no carry CALL
CPE	aa	354	EC	11/17	If even parity CALL
CPO	aa	344	E4	11/17	If odd parity CALL

Mnemonics		oct	hex	States	Description
CMA		057	2F	4	Compliment A
CMC		077	3F	4	Compliment carry
CMP	A	277	BF	4	Clear flags
CMP	B	270	B8	4	Compare A with B
CMP	C	271	B9	4	Compare A with C
CMP	D	272	BA	4	Compare A with D
CMP	E	273	BB	4	Compare A with E
CMP	H	274	BC	4	Compare A with H
CMP	L	275	BD	4	Compare A with L
CMP	M	276	BE	7	Compare A with LOC(H&L)
CPI	v	376	FE	7	Compare A with v
DAA		047	27	4	Decimal adjust A
DAD	B	011	09	10	Add B&C to H&L
DAD	D	031	19	10	Add D&E to H&L
DAD	H	051	29	18	Double H&L (shift H&L left)
DAD	SP	071	39	10	Add SP to H&L
DCR	A	075	3D	5	Decrement A
DCR	B	005	05	5	Decrement B
DCR	C	015	0D	5	Decrement C
DCR	D	025	15	5	Decrement D
DCR	E	035	1D	5	Decrement E
DCR	H	045	25	5	Decrement H
DCR	L	055	2D	5	Decrement L
DCR	M	065	35	10	Decrement LOC(H&L)
DCX	B	013	0B	5	Decrement B&C
DCX	D	033	1B	5	Decrement D&E
DCX	H	053	2B	5	Decrement H&L
DCX	SP	073	3B	5	Decrement SP
DI		363	F3	4	Disable interrupts
EI		373	FB	4	Enable interrupts
HLT		166	76	7	halt until interrupt
IN	v	333	DB	10	Input from device v
INR	A	074	3C	5	Increment A
INR	B	004	04	5	Increment B
INR	C	014	0C	5	Increment C

Mnemonics		oct	hex	States	Description
INR	D	024	14	5	Increment D
INR	E	034	1C	5	Increment E
INR	H	044	24	5	Increment H
INR	L	054	2C	5	Increment L
INR	M	064	34	10	Increment LOC(H&L)
INX	B	003	03	5	Increment B&C
INX	D	023	13	5	Increment D&E
INX	H	043	23	5	Increment H&L
INX	SP	063	33	5	Increment SP
JMP	aa	303	C3	10	Jump to LOC aa
JZ	aa	312	CA	10	If zero JMP
JNZ	aa	302	C2	10	If not zero JMP
JP	aa	362	F2	10	If plus JMP
JM	aa	372	FA	10	If minus JMP
JC	aa	332	DA	10	If carry JMP
JNC	aa	322	D2	10	If no carry JMP
JPE	aa	352	EA	10	If even parity JMP
JPO	aa	342	E2	10	If odd parity JMP
LDA	aa	072	3A	13	Load A from LOC aa
LDAX	B	012	0A	7	Load A from LOC(B&C
LDAX	D	032	1A	7	Load A from LOC(D&E)
LHLD	aa	052	2A	16	Load H&L from LOC aa
LXI	B,vv	001	01	10	Load B&C with vv
LXI	D,vv	021	11	10	Load D&E with vv
LXI	H,vv	041	21	10	Load H&L with vv
LXI	SP,vv	061	31	10	Load SP with vv
MOV	A,B	170	78	5	Move B to A
MOV	A,C	171	79	5	Move C to A
MOV	A,D	172	7A	5	Move D to A
MOV	A,E	173	7B	5	Move E to A
MOV	A,H	174	7C	5	Move H to A
MOV	A,L	175	7D	5	Move L to A
MOV	A,M	176	7E	7	Move LOC(H&L) to A
MOV	B,A	107	47	5	Move A to B
MOV	B,C	101	41	5	Move C to B
MOV	B,D	102	42	5	Move D to B

Mnemonics		oct	hex	States	Description
MOV	B,E	103	43	5	Move E to B
MOV	B,H	104	44	5	Move H to B
MOV	B,L	105	45	5	Move L to B
MOV	B,M	106	46	7	Move LOC(H&L) to B
MOV	C,A	117	4F	5	Move A to C
MOV	C,B	110	48	5	Move B to C
MOV	C,D	112	4A	5	Move D to C
MOV	C,E	113	4B	5	Move E to C
MOV	C,H	114	4C	5	Move H to C
MOV	C,L	115	4D	5	Move L to C
MOV	C,M	116	4E	7	Move LOC(H&L) to C
MOV	D,A	127	57	5	Move A to D
MOV	D,B	120	50	5	Move B to D
MOV	D,C	121	51	5	Move C to D
MOV	D,E	123	53	5	Move E to D
MOV	D,H	124	54	5	Move H to D
MOV	D,L	125	55	5	Move L to D
MOV	D,M	126	56	7	Move LOC(H&L) to D
MOV	E,A	137	5F	5	Move A to E
MOV	E,B	130	58	5	Move B to E
MOV	E,C	131	59	5	Move C to E
MOV	E,D	132	5A	5	Move D to E
MOV	E,H	134	5C	5	Move H to E
MOV	E,L	135	5D	5	Move L to E
MOV	E,M	136	5E	7	Move LOC(H&L) to E
MOV	H,A	147	67	5	Move A to H
MOV	H,B	140	60	5	Move B to H
MOV	H,C	141	61	5	Move C to H
MOV	H,D	142	62	5	Move D to H
MOV	H,E	143	63	5	Move E to H
MOV	H,L	145	65	5	Move L to H
MOV	H,M	146	66	7	Move LOC(H&L) to H
MOV	L,A	157	6F	5	Move A to L
MOV	L,B	150	68	5	Move B to L
MOV	L,C	151	69	5	Move C to L
MOV	L,D	152	6A	5	Move D to L
MOV	L,E	153	6B	5	Move E to L
MOV	L,H	154	6C	5	Move H to L
MOV	L,M	156	6E	7	Move LOC(H&L) to L
MOV	M,A	167	77	7	Move A to LOG(H&L)

Mnemonics		oct	hex	States	Description
MOV	M,B	160	70	7	Move B to LOC(H&L)
MOV	M,C	161	71	7	Move C to LOC(H&L)
MOV	M,D	162	72	7	Move D to LOC(H&L)
MOV	M,E	163	73	7	Move E to LOC(H&L)
MOV	M,H	164	74	7	Move H to LOC(H&L)
MOV	M,L	165	75	7	Move L to LOC(H&L)
MVI	A,v	076	3E	7	Move v to A
MVI	B,v	006	06	7	Move v to B
MVI	C,v	016	OE	7	Move v to C
MVI	D,v	026	16	7	Move v to D
MVI	E,v	036	1E	7	Move v to E
MVI	H,v	046	26	7	Move v to H
MVI	L,v	056	2E	7	Move v to L
MVI	M,v	066	36	10	Move v to LOC(H&L)
NOP		000	00	4	No operation
ORA	A	267	B7	4	Test A and clear away
ORA	B	260	B0	4	OR B with A
ORA	C	261	B1	4	OR C with A
ORA	D	262	B2	4	OR D with A
ORA	E	263	B3	4	OR E with A
ORA	H	264	B4	4	OR H with A
ORA	L	265	B5	4	OR L with A
ORA	M	266	B6	7	OR LOC(H&L) with A
ORI	v	366	F6	7	OR v with A
OUT	v	323	D3	10	Output to device v
PCHL		351	E9	5	Jump to LOC(H&L)
POP	B	301	C1	10	Pop B&C from stack
POP	D	321	D1	10	Pop D&E from stack
POP	H	341	E1	10	Pop H&L from stack
POP	PSW	361	F1	10	Pop A and flags from stack
PUSH	B	305	C5	11	Push B&C onto stack
PUSH	D	325	D5	11	Push D&E onto stack
PUSH	H	345	E5	11	Push H&L onto stack
PUSH	PSW	365	F5	11	Push A and flags onto stack
RAL		027	17	4	Rotate A left through carry

Mnemonics		oct	hex	States	Description
RAR		037	1F	4	Rotate A right through carry
RLC		007	07	4	Rotate A left
RRC		017	0F	4	Rotate A right
RET		311	C9	10	Return from subroutine
RZ		310	C8	5/11	If zero RET
RNZ		300	C0	5/11	If not zero RET
RP		360	F0	5/11	If plus RET
RM		370	F8	5/11	If minus RET
RC		330	D8	5/11	If carry RET
RNC		320	D0	5/11	If no carry RET
RPE		350	E8	5/11	If even parity RET
RPO		340	E0	5/11	If odd parity RET
RST	0	307	C7	11	Restart subroutine 000
RST	1	317	CF	11	Restart subroutine 010
RST	2	327	D7	11	Restart subrountine 020
RST	3	337	DF	11	Restart subroutine 030
RST	4	347	E7	11	Restart subroutine 040
RST	5	357	EF	11	Restart subroutine 050
RST	6	367	F7	11	Restart subroutine 060
RST	7	377	FF	11	Restart subroutine 070
SPHL		371	F9	5	Load SP from H&L
SHLD	aa	042	22	16	Store H&L at LOC aa
STA	aa	062	32	13	Store A at LOC aa
STAX	B	002	02	7	Store A at LOC(B&C)
STAX	D	022	12	7	Store A at LOC(D&E)
STC		067	37	4	Set carry
SUB	A	227	97	4	Clear A and flags
SUB	B	220	90	4	Subtract B from A
SUB	C	221	91	4	Subtract C from A

Mnemonics		oct	hex	States	Description
SUB	D	222	92	4	Subtract D from A
SUB	E	223	93	4	Subtract E from A
SUB	H	224	94	4	Subtract H from A
SUB	L	225	95	4	Subtract L from A
SUB	M	226	96	7	Subtract LOC(H&L) from A
SUI	v	326	D6	7	Subtract v from A
SBB	A	237	9F	4	Set A to minus carry
SBB	B	230	9B	4	Subtract B and borrow from A
SBB	C	231	99	4	Subtract C and borrow from A
SBB	D	232	9A	4	Subtract D and borrow from A
SBB	E	233	9B	4	Subtract E and borrow from A
SBB	H	234	9C	4	Subtract H and borrow from A
SBB	L	235	9D	4	Subtract L and borrow from A
SBB	M	236	9E	7	Subtract LOC(H&L) and borrow from A
SBI	vv	336	DE	7	Subtract v and borrow from A
XCHG		358	EB	4	Exchange D&E with H&L
XTHL		343	E3	18	Exchange top of stack with H&L
XRA	A	257	AF	4	Clear A and flags
XRA	B	250	A8	4	Exclusive OR B with A
XRA	C	251	A9	4	Exclusive OR C with A
XRA	D	252	AA	4	Exclusive OR D with A
XRA	E	253	AB	4	Exclusive OR E with A
XRA	H	254	AC	4	Exclusive OR H with A
XRA	L	255	AD	4	Exclusive OR L with A

Mnemonics		oct	hex	States	Description
XRA	M	256	AE	7	Exclusive OR LOC(H&L) with A
XRI	v	356	EE	7	Exclusive OR v with A

Status Flags

flag	description	flag = 1	flag = 0
Z	Result is zero	Zero (Z)	Not Zero (NZ)
S	Sign of Result (MSB)	Minus (M)	Plus (P)
P	Parity of Result	Even (PE)	Odd (PO)
C	Carry from Result	Carry (C)	No Carry (NC)
AC	Auxiliary Carry from bit 4 of Result	Carry	No Carry

Instructions Affecting Flags

ACI all flags
ADC all flags
ADD all flags
ADI all flags
ANA all flags, C = AC = 0
ANI all flags, C = AC = 0
CMC only C flag, C = 1 – C
CMP all flags
CPI all flags
DAA all flags
DAD only C flag
DCR all but C flag
INR all but C flag
ORA all flags, C = AC = 0
ORI all flags, C = AC = 0
POP PSW all flags
RAL only C flag
RAR only C flag
RLC only C flag
RRC only C flag
SBB all flags
SBI all flags
STC only C flag, C = 1
SUB all flags
SUI all flags
XRA all flags, C = AC = 0
XRI all flags, C = AC = 0

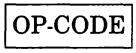
Compare Instruction

Result of CMP 8 instruction

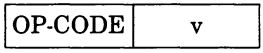
A $<$ B Carry
A = B Zero
A $>$ B Not Zero And No Carry
A $\geq$ B No Carry
A $\neq$ B Not Zero
A $\leq$ B Zero or Carry

Instruction Formats

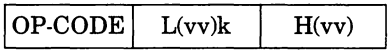
One Byte Instructions

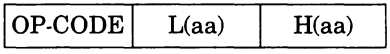


Two Byte Instructions



Three Byte Instructions





Notation

Symbol	Description
A	Accumulator (8-bits)
B, C, D, E, H, L	Work Registers (8-bits)
B&C, D&E, H&L	Register Pairs (16-bits)
M	Contents of memory location pointed to by H&L
PC	Program Counter
SP	Stack Pointer
PSW	Processor Status Word
rp	Register Pair (B&C, D&E or H&L)
v	8 bit value after instruction
vv	16 bit value after instruction
aa	16 bit address after instruction
H(vv)	Most significant byte of vv
L(vv)	Least significant byte of vv
LOC aa	Contents of memory location aa
LOC(rp)	Contents of memory location pointed to by register pair rp

ASCII Control Characters

char	key	description
ACK	CTL-F	Acknowledge
BEL	CTL-G	Bell

BS	CTL-H	Backspace
CAN	CTL-X	Cancel
CR	CTL-M	Carriage Return
DC1	CTL-Q	Device Control 1 (X-ON)
DC2	CTL-R	Device Control 2
DC3	CTL-S	Device Control 3 (X-OFF)
DC4	CTL-T	Device Control 4
DEL		Delete
DLE	CTL-P	Data Link Escape
EM	CTL-Y	End of Medium
ENQ	CTL-E	Enquiry
EOT	CTL-D	End of Transmission
ESC		Escape
ETB	CTL-W	End of Transmission Block
ETX	CTL-C	End of Text
FF	CTL-L	Form Feed
FS		File Separator
GS		Group Separator
HT	CTL-I	Horizontal Tab
LF	CTL-J	Line Feed
NAK	CTL-U	Negative Acknowledge
NUL		Null
RS		Record Separator
SI	CTL-O	Shift In
SO	CTL-N	Shift Out
SOH	CTL-A	Start of Heading
SP		Space (a blank)
STX	CTL-B	Start of Text
SUB	CTL-Z	Substitute
SYN	CTL-V	Synchronous Idle
US		Unit Separator
VT	CTL-K	Vertical Tab

DEC	HEX	OCT	CHAR	PARITY	MNEMONIC	CYCLES	OCT
0	00	000	NUL	EVEN	NOP	1	000
1	01	001	SOH	ODD	LXI B,vv	3	001
2	02	002	STX	ODD	STAX B	2	002
3	03	003	ETX	EVEN	INX B	1	003
4	04	004	EOT	ODD	INR B	1	004
5	05	005	ENQ	EVEN	DCR B	1	005
6	06	006	ACK	EVEN	MVI B,v	2	006
7	07	007	BEL	ODD	RLC	1	007
8	08	010	BS	ODD			010
9	09	011	HT	EVEN	DAD B	3	011
10	0A	012	LF	EVEN	LDAX B	2	012

DEC	HEX	OCT	CHAR	PARITY	MNEMONIC	CYCLES	OCT
11	0B	013	VT	ODD	DCX B	1	013
12	0C	014	FF	EVEN	INR C	1	014
13	0D	015	CR	ODD	DCR C	1	015
14	0E	016	SO	ODD	MVI C,v	2	016
15	0F	017	SI	EVEN	RRC	1	017
16	10	020	DLE	ODD			020
17	11	021	DC1	EVEN	LXI C,vv	3	021
18	12	022	DC2	EVEN	STAX D	2	022
19	13	023	DC3	ODD	INX D	1	023
20	14	024	DC4	EVEN	INR D	1	024
21	15	025	NAK	ODD	DCR D	1	025
22	16	026	SYN	ODD	MVI D,v	2	026
23	17	027	ETB	EVEN	RAL	1	027
24	18	030	CAN	EVEN			030
25	19	031	EM	ODD	DAD D	3	031
26	1A	032	SUB	ODD	LDAX D	2	032
27	1B	033	ESC	EVEN	DCX D	1	033
28	1C	034	FS	ODD	INR E	1	034
29	1D	035	GS	EVEN	DCR E	1	035
30	1E	036	RS	EVEN	MVI E,v	2	036
31	1F	037	US	ODD	RAR	1	037
32	20	040	SP	ODD			040
33	21	041	!	EVEN	LXI H,vv	3	041
34	22	042	"	EVEN	SHLD aa	5	042
35	23	043	#	ODD	INX H	1	043
36	24	044	$	EVEN	INR H	1	044
37	25	045	%	ODD	CR H	1	045
38	26	046	&	ODD	MVI H,v	2	046
39	27	047	'	EVEN	DAA	1	047
40	28	050	(	EVEN			050
41	29	051	)	ODD	DAD H	3	051
42	2A	052	*	ODD	LHLD aa	5	052
43	2B	053	+	EVEN	DCX H	1	053
44	2C	054	,	ODD	INR L	1	054
45	2D	055	-	EVEN	DCR L	1	055
46	2E	056	.	EVEN	MVI L,v	2	056
47	2F	057	/	ODD	CMA	1	057
48	30	060	0	EVEN			060
49	31	061	1	ODD	LXI SP,vv	3	061
50	32	062	2	ODD	STA aa	4	062
51	33	063	3	EVEN	INX SP	1	063
52	34	064	4	ODD	INR M	3	064
53	35	065	5	EVEN	DCR M	3	065
54	36	066	6	EVEN	MVI M,v	3	066
55	37	067	7	ODD	STC	1	067
56	38	070	8	ODD			070
57	39	071	9	EVEN	DAD SP	3	071

DEC	HEX	OCT	CHAR	PARITY	MNEMONIC	CYCLES	OCT
58	3A	072	:	EVEN	LDA aa	4	072
59	3B	073	;	ODD	DCX SP	1	073
60	3C	074	<	EVEN	INR A	1	074
61	3D	075	=	ODD	DCR A	1	075
62	3E	076	>	ODD	MVI A,v	2	076
63	3F	077	?	EVEN	CMC	1	077
64	40	100	@	ODD	MOV B,B	1	100
65	41	101	A	EVEN	MOV B,C	1	101
66	42	102	B	EVEN	MOV B,D	1	102
67	43	103	C	ODD	MOV B,E	1	103
68	44	104	D	EVEN	MOV B,H	1	104
69	45	105	E	ODD	MOV B,L	1	105
70	46	106	F	ODD	MOV B,M	2	106
71	47	107	G	EVEN	MOV B,A	1	107
72	48	110	H	EVEN	MOV C,B	1	110
73	49	111	I	ODD	MOV C,C	1	111
74	4A	112	J	ODD	MOV C,D	1	112
75	4B	113	K	EVEN	MOV C,E	1	113
76	4C	114	L	ODD	MOV C,H	1	114
77	4D	115	M	EVEN	MOV C,L	1	115
78	4E	116	N	EVEN	MOV CM,	2	116
79	4F	117	O	ODD	MOV C,A	1	117
80	50	120	P	EVEN	MOV D,B	1	120
81	51	121	Q	ODD	MOV D,C	1	121
82	52	122	R	ODD	MOV D,D	1	122
83	53	123	S	EVEN	MOV D,E	1	123
84	54	124	T	ODD	MOV D,H	1	124
85	55	125	U	EVEN	MOV D,L	1.	125
86	56	126	V	EVEN	MOV D,M	2	126
87	57	127	W	ODD	MOV D,A	1	127
88	58	130	X	ODD	MOV E,B	1	130
89	59	131	Y	EVEN	MOV E,C	1	131
90	5A	132	Z	EVEN	MOV E,D	1	132
91	5B	133	[	ODD	MOV E,E	1	133
92	5C	134	\	EVEN	MOV E,H	1	134
93	5D	135	]	ODD	MOV E,L	1	135
94	5E	136	↑	ODD	MOV E,M	2	136
95	5F	137	←	EVEN	MOV E,A	1	137
96	60	140	'	EVEN	MOV H,B	1	140
97	61	141	a	ODD	MOV H,C	1	141
98	62	142	b	ODD	MOV H,D	1	142
99	63	143	c	EVEN	MOV H,E	1	143
100	64	144	d	ODD	MOV H,H	1	144
101	65	145	e	EVEN	MOV H,L	1	145
102	66	146	f	EVEN	MOV H,M	2	146
103	67	147	g	ODD	MOV H,A	1	147

DEC	HEX	OCT	CHAR	PARITY	MNEMONIC	CYCLES	OCT
104	68	150	h	ODD	MOV L,B	1	150
105	69	151	i	EVEN	MOV L,C	1	151
106	6A	152	j	EVEN	MOV L,D	1	152
107	6B	153	k	ODD	MOV L,E	1	153
108	6C	154	l	EVEN	MOV L,H	1	154
109	6D	155	m	ODD	MOV L,L	1	155
110	6E	156	n	ODD	MOV L,M	1	156
111	6F	157	o	EVEN	MOV L,A	1	157
112	70	160	p	ODD	MOV M,B	2	160
113	71	161	q	EVEN	MOV M,C	2	161
114	72	162	r	EVEN	MOV M,D	2	162
115	73	163	s	ODD	MOV M,E	2	163
116	74	164	t	EVEN	MOV M,H	2	164
117	75	165	u	ODD	MOV M,L	2	165
118	76	166	v	ODD	HLT	2	166
119	77	167	w	EVEN	MOV M,A	2	167
120	78	170	x	EVEN	MOV A,B	1	170
121	79	171	y	ODD	MOV A,C	1	171
122	7A	172	z	ODD	MOV A,D	1	172
123	7B	173	{	EVEN	MOV A,E	1	173
124	7C	174	|	ODD	MOV A,H	1	174
125	7D	175	}	EVEN	MOV A,L	1	175
126	7E	176	~	EVEN	MOV A,M	2	176
127	7F	177	DEL	ODD	MOV A,A	1	177
128	80	200	NUL	ODD	ADD B	1	200
129	81	201	SOH	EVEN	ADD C	1	201
130	82	202	STX	EVEN	ADD D	1	202
131	83	203	ETX	ODD	ADD E	1	203
132	84	204	EOT	EVEN	ADD H	1	204
133	85	205	ENQ	ODD	ADD L	1	205
134	86	206	ACK	ODD	ADD M	2	206
135	87	207	BEL	EVEN	ADD A	1	207
136	88	210	BS	EVEN	ADC B	1	210
137	89	211	HT	ODD	ADC C	1	211
138	8A	212	LF	ODD	ADC D	1	212
139	8B	213	VT	EVEN	ADC E	1	213
140	8C	214	FF	ODD	ADC H	1	214
141	8D	215	CR	EVEN	ADC L	1	215
142	8E	216	SO	EVEN	ADC M	2	216
143	8F	217	SI	ODD	ADC A	1	217
144	90	220	DLE	EVEN	SUB B	1	220
145	91	221	DC1	ODD	SUB C	1	221
146	92	222	DC2	ODD	SUB D	1	222
147	93	223	DC3	EVEN	SUB E	1	223
148	94	224	DC4	ODD	SUB H	1	224
149	95	225	NAK	EVEN	SUB L	1	225
150	96	226	SYN	EVEN	SUB M	2	226

DEC	HEX	OCT	CHAR	PARITY	MNEMONIC	CYCLES	OCT
151	97	227	ETB	ODD	SUB A	1	227
152	98	230	CAN	ODD	SBB B	1	230
153	99	231	EM	EVEN	SBB C	1	231
154	9A	232	SUB	EVEN	SBB D	1	232
155	9B	233	ESC	ODD	SBB E	1	233
156	9C	234	FS	EVEN	SBB H	1	234
157	9D	235	GS	ODD	SBB L	1	235
158	9E	236	RS	ODD	SBB M	2	236
159	9F	237	US	EVEN	SBB A	1	237
160	A0	240	SP	EVEN	ANA B	1	240
161	A1	241	!	ODD	ANA C	1	241
162	A2	242	"	ODD	ANA D	1	242
163	A3	243	#	EVEN	ANA E	1	243
164	A4	244	$	ODD	ANA H	1	244
165	A5	245	%	EVEN	ANA L	1	245
166	A6	246	&	EVEN	ANA M	2	246
167	A7	247	'	ODD	ANA A	1	247
168	A8	250	(	ODD	XRA B	1	250
169	A9	251	)	EVEN	XRA C	1	251
170	AA	252	*	EVEN	XRA D	1	252
171	AB	253	+	ODD	XRA E	1	253
172	AC	254	,	EVEN	XRA H	1	254
173	AD	255	-	ODD	XRA L	1	255
174	AE	256	.	ODD	XRA M	2	256
175	AF	257	/	EVEN	XRA A	1	257
176	B0	260	0	ODD	ORA B	1	260
177	B1	261	1	EVEN	ORA C	1	261
178	B2	262	2	EVEN	ORA D	1	262
179	B3	263	3	ODD	ORA E	1	263
180	B4	264	4	EVEN	ORA H	1	264
181	B5	265	5	ODD	ORA L	1	265
182	B6	266	6	ODD	ORA M	2	266
183	B7	267	7	EVEN	ORA A	1	267
184	B8	270	8	EVEN	CMP B	1	270
185	B9	271	9	ODD	CMP C	1	271
186	BA	272	:	ODD	CMP D	1	272
187	BB	273	;	EVEN	CMP E	1	273
188	BC	274	<	ODD	CMP H	1	274
189	BD	275	=	EVEN	CMP L	1	275
190	BE	276	>	EVEN	CMP M	2	276
191	BF	277	?	ODD	CMP A	1	277
192	C0	300	@	EVEN	RNZ	1/3	300
193	C1	301	A	ODD	POP B	3	301
194	C2	302	B	ODD	JNZ aa	3	302
195	C3	303	C	EVEN	JMP aa	3	303
196	C4	304	D	ODD	CNZ aa	3/5	304

DEC	HEX	OCT	CHAR	PARITY	MNEMONIC	CYCLES	OCT
197	C5	305	E	EVEN	PUSH B	3	305
198	C6	306	F	EVEN	ADI v	2	306
199	C7	307	G	ODD	RST O	3	307
200	C8	310	H	ODD	RZ	1/3	310
201	C9	311	I	EVEN	RET	3	311
202	CA	312	J	EVEN	JZ aa	3	312
203	CB	313	K	ODD			313
204	CC	314	L	EVEN	CZ aa	3/5	314
205	CD	315	M	ODD	CALL aa	5	315
206	CE	316	N	ODD	ACI v	2	316
207	CF	317	O	EVEN	RST 1	3	317
208	D0	320	P	ODD	RNC	1/3	320
209	D1	321	Q	EVEN	POP D	3	321
210	D2	322	R	EVEN	JNC aa	3	322
211	D3	323	S	ODD	OUT v	3	323
212	D4	324	T	EVEN	CNC aa	3/5	324
213	D5	325	U	ODD	PUSH D	3	325
214	D6	326	V	ODD	SUI v	2	326
215	D7	327	W	EVEN	RST 2	3	327
216	D8	330	X	EVEN	RC	1/3	330
217	D9	331	Y	ODD			331
218	DA	332	Z	ODD	JC aa	3	332
219	DB	333	[	EVEN	IN v	3	333
220	DC	334	\	ODD	CC aa	3/5	334
221	DD	335	]	EVEN			335
222	DE	336	↑	EVEN	SBI v	2	336
223	DF	337	←	ODD	RST 3	3	337
224	E0	340	'	ODD	RPO	1/3	340
225	E1	341	a	EVEN	POP H	3	341
226	E2	342	b	EVEN	JPO aa	3	342
227	E3	343	c	ODD	XTHL	5	343
228	E4	344	d	EVEN	CPO aa	3/5	344
229	E5	345	e	ODD	PUSH H	3	345
230	E6	346	f	ODD	ANI v	2	346
231	E7	347	g	EVEN	RST 4	3	347
232	E8	350	h	EVEN	RPE	1/3	350
233	E9	351	i	ODD	PCHL	1	351
234	EA	352	j	ODD	JPE aa	3	352
235	EB	353	k	EVEN	XCHG	1	353
236	EC	354	l	ODD	CPE aa	3/5	354
237	ED	355	m	EVEN			355
238	EE	356	n	EVEN	XRI v	2	356
239	EF	357	o	ODD	RST 5	3	357
240	F0	360	p	EVEN	RP	1/3	360
241	F1	361	q	ODD	POP PSW	3	361

DEC	HEX	OCT	CHAR	PARITY	MNEMONIC	CYCLES	OCT
242	F2	362	r	ODD	JP aa	3	362
243	F3	363	s	EVEN	DI	1	363
244	F4	364	t	ODD	CP aa	3/5	364
245	F5	365	u	EVEN	PUSH PSW	3	365
246	F6	366	v	EVEN	ORI v	2	366
247	F7	367	w	ODD	RST 6	3	367
248	F8	370	x	ODD	RM	1/3	370
249	F9	371	y	EVEN	SPHL	1	371
250	FA	372	z	EVEN	JM aa	3	372
251	FB	373	{	ODD	EI	1	373
252	FC	374	|	EVEN	CM aa	3/5	374
253	FD	375	}	ODD			375
254	FE	376	~	ODD	CPI v	2	376
255	FF	377	DEL	EVEN	RST 7	3	377

Appendix I

Sample Assembly Listing

NOTE: The following is an example assembly language program to show BDOS calls, and is reprinted with the permission of Tim Pugh. It is taken from the CP/M User Group #25. It does **not** support X-on and X-off and is not recommended for file transfer for this reason. It is shown only as an example of BDOS calls. For file transfers using modems, use later programs by Tim Pugh, MODEM7 in User Group #47, or any of the various commercial products.

```
;************************************************************************
;
; TITLE: 88-MODEM DRIVER ROUTINE WITH DISK FILE TRANSFER
;
; VERSION: 1.00                LAST REVISION: 7/7/78
;
; AUTHOR: TIM PUGH
;         9655-M HOMESTEAD COURT
;         LAUREL, MD. 20810
;         (301) 776-5253
;
;****************** SUMMARY OF CONTROL CHARACTERS *********************
;
;CONTROL CHARACTERS ARE INDICATED BY A "^" PRECEEDING THE CHARACTER.
;
; ^A    ASSIGN CONSOLE DEVICE
; ^B    TRANSMIT A "BREAK"
; ^D    DISCONTINUE CURRENT MODE
; ^F    FILE NAME SPECIFIED FOR SUBSEQUENT RX/TX
; ^R    RECEIVE (RX) A DISK FILE FROM A REMOTE DEVICE
; ^T    TRANSMIT (TX) A DISK FILE TO A REMOTE DEVICE
; ^W    ASCII ETB -- MUST BE THE LAST CHARACTER OF A TX FILE
;
;******************* OPERATING INSTRUCTIONS ***************************
;
;THIS PROGRAM IS DESIGNED TO OPERATE UNDER CP/M.  IF A FILE TRANSFER
;IS TO BE PERFORMED, THE FILE NAME SHOULD BE SPECIFIED WHEN THE PROGRAM
;IS CALLED, E.G.
;                    88-MODEM  <FILENAME>.<FILETYPE>
;
;FOLLOWING THE PROGRAM SIGN ON MESSAGE, THE SYSTEM WAITS FOR OPERATOR
;INPUT.  AT THIS POINT THE OPERATOR CAN INITIATE HANDSHAKING (AS DES-
;CRIBED UNDER "FILE TX MODE" BELOW) BY ENTERING AN "H".  ALTERNATIVELY,
;THE OPERATOR CAN SELECT FROM ONE OF 6 BAUD RATES BY ENTERING A "B".
;FINALLY, HE CAN DEFAULT TO 300 BAUD AND NO HAND-SHAKE BY ENTERING
;A "CR" (CARRIAGE RETURN) AS THE INITIAL ENTRY.
;
;NEXT, THE OPERATOR IS DIRECTED TO SELECT THE MODE OF OPERATION BY ENTERING
;EITHER AN "O" FOR ORIGINATE OR AN "A" FOR ANSWER.  IF ANSWER MODE IS SELECTED,
;THE PROGRAM INITIALIZES THE SYSTEM ACCORDINGLY AND WAITS FOR RING DETECT.
;THE PHONE WILL BE ANSWERED ON THE FIRST RING AND GENERATES THE 2225 HZ
;CARRIER SIGNAL.  UPON DETECTION OF THE RESPONDING 1270 HZ TONE, A SIGN-ON
;MESSAGE IS TRANSMITTED TO THE REMOTE DEVICE AND A LOCAL MESSAGE DIRECTING
;THE COMMENCEMENT OF NORMAL TRANSMISSION IS DISPLAYED ON THE CONSOLE.
;IF THE REMOTE DEVICE IS A TERMINAL IT SHOUD BE IN HALF-DUPLEX MODE.
;OF THE ABOVE MENTIONED CONTROL CHARACTERS, ONLY ^T, ^R, AND ^D ARE
;SUPPORTED AT THE REMOTE DEVICE.  HOWEVER, ALL ARE SUPPORTED AT THE LOCAL
;CONSOLE.  IF THE LOCAL CONSOLE IS UNATTENDED, A FILE TRANSFER CAN BE
;INITIATED REMOTELY PROVIDED THE FILE NAME WAS PREVIOUSLY SPECIFIED.
```

```
;COMMUNICATIONS CAN BE TERMINATED REMOTELY SIMPLY BY HANGING UP THE PHONE.
;THIS CAUSES THE SYSTEM TO RETURN TO INITIALIZATION OF ANSWER MODE AND
;AWAIT THE NEXT REMOTE CALL.  ALTERNATIVELY, A "^D" ISSUED REMOTELY WILL
;TERMINATE COMMUNICATIONS AND TAKE THE SYSTEM OUT OF ANSWER MODE.  THIS
;MEANS THAT THE NEXT CALLER WON'T BE GREETED BY A PERHAPS UNEXPECTED
;2225HZ TONE.
;
;IF ORIGINATE MODE IS SELECTED, THE SYSTEM WAITS FOR DIAL TONE DETECTION.
;ONCE THE DIAL TONE HAS BEEN DETECTED - ANNOUNCED ON THE CONSOLE - DIALING
;CAN BEGIN.  CONVENTIONAL DIALING IS PERFORMED BY SIMPLY ENTERING THE
;NUMBER TO BE DIALED AT THE CONSOLE KEYBOARD.  AS EACH NUMBER IS DIALED
;A DASH IS DISPLAYED.  IT IS POSSIBLE TO EXCEED THE DIALING SPEED OF THE
;DEVICE.  THIS IS INDICATED BY THE ABSENCE OF A DASH BETWEEN DIGITS.
;
;BOTH AUTODIALING AND ROTARY DIALING ARE SUPPORTED.  AUTODIALING IS INIT-
;IATED BY ENTERING ONE OF 4 ALPHA CHARACTERS AS THE FIRST "DIGIT" DIALED.
;THE CHARACTERS SUPPORTED ARE "H,M,Z,W".  OTHER CHARACTERS MAY BE SELECTED
;PROVIDED THE 2 LSB'S OF THE ASCII CODE FOR THE SET SELECTED COVER THE
;RANGE OF 00,01,10,11.  THE PHONE NUMBERS ASSIGNED TO THESE CHARACTERS
;CAN BE FOUND IN THE LISTING JUST PRIOR THE THE "MESSAGES" SECTION.
;ROTARY DIALING IS SUPPORTED FOR THE CHARACTER "M" WHEREBY UP TO 4
;NUMBERS WILL BE DIALED IN SEQUENCE IF EACH PRECEEDING NUMBER IS FOUND
;BUSY.  THE SIZE OF THE ROTARY GROUP IS NOT RESTRICTED TO 4.  ANY NUMBER
;GREATER THAN OR EQUAL TO 2 CAN BE SUPPORTED.
;
;AFTER DIALING IS COMPLETED AND THE REMOTE CARRIER HAS BEEN DETECTED,
;THE SYSTEM IS PLACED IN THE "CHARACTER TX/RX MODE".  THIS IS THE MODE
;USED FOR KEYBOARD COMMUNICATIONS WITH THE REMOTE DEVICE.  WHILE IN
;THIS MODE, A BREAK MAY BE TRANSMITTED BY ENTERING A "^B".  IF
;MORE THAN ONE FILE IS TO BE TRANSFERRED, THE NEW FILE NAME CAN BE
;SPECIFIED BY ENTERING A "^F".  CONSOLE MESSAGES WILL PROMPT THE
;OPERATOR FOR APPROPRIATE ENTRIES.  IT IS POSSIBLE TO REASSIGN THE
;CONSOLE DEVICE BY ENTERING A "^A".  HERE TOO, CONSOLE MESSAGES WILL
;PROMPT THE OPERATOR FOR APPROPRIATE ENTRIES.  THIS FEATURE IS ONLY
;SUPPORTED ON SYSTEMS HAVING A TDL "ZAPPLE" MONITOR.
;
;IN ORDER TO INITIATE THE "FILE TX MODE" THE USER SHOULD ENTER A "^T".
;THE DATA IS DISPLAYED ON THE CONSOLE DEVICE AS IT IS TRANSMITTED.  UPON
;COMPLETION OF FILE TRANSMISSION, THE SYSTEM REVERTS TO THE CHARACTER
;MODE.  RETURN TO CHARACTER MODE PRIOR TO COMPLETION CAN BE FORCED BY
;ENTERING "^D".  "HAND-SHAKING" AS REQUIRED BY SOME UNIVAC 1100 SERIES
;COMPUTERS HAS BEEN INCLUDED.  TO ENABLE THIS, ENTER A "H" AS THE
;DIALING INITIATION CHARACTER (AS THE CONSOLE MESSAGE DIRECTS).
;THE END OF FILE IN THIS MODE IS "^W".  THUS, THE FILE TO BE TRANS-
;MITTED MUST HAVE A "^W" WRITTEN AS THE LAST CHARACTER ON THE FILE.
;
;TO INITIATE THE "FILE RX MODE" THE USER ENTERS A "^R".  OPERATION IS
;SIMILAR TO FILE TX MODE DESCRIBED ABOVE.  SINCE CHARACTER TRANSMISSION
;IS SUPPORTED IN THIS MODE, THE USER SHOULD SET UP FOR FILE TRANSFER
```

```
;BY ISSUEING A COMMAND TO THE REMOTE DEVICE EXCEPT FOR THE CARRIAGE
;RETURN.  TYPE "^R" AND AFTER THE READY MESSAGE IS GIVEN, TYPE "CR".
;THE CR WILL GO TO THE REMOTE DEVICE AND INITIATE TRANSMISSION.
;THIS MODE DOES NOT TERMINATE AUTOMATICALLY.  A "^D" MUST BE TYPED
;IN ORDER TO RETURN TO THE CHARACTER MODE.
;
;ESCAPE FROM THE CHARACTER MODE TO THE INITIAL DIALING SEQUENCE IS
;MADE BY ENTERING A "^D".
;
;RETURN TO CP/M IS EFFECTED BY ENTERING A "^D" WHILE IN THE "INITIAL
;DIALING SEQUENCE MODE".
;
;********************* MODEM CONTROL EQUATES *************************
;
;--I/O ADDRESSES
;
MODEM0  EQU     0C0H    ;UART STATUS/CONTROL
MODEM1  EQU     0C1H    ;DATA IN/OUT
MODEM2  EQU     0C2H    ;FILTER & CONTROL WORD SELECT
MODEM3  EQU     0C3H    ;(MODEM,DIALER,BAUD-RATE) CONTROL
;
;--UART/MODEM STATUS (IN MODEM0)
;
TBE     EQU     001H    ;TRANSMIT BUFFER EMPTY
RDA     EQU     002H    ;RECEIVE DATA AVAILABLE
DBY     EQU     004H    ;DIALER BUSY
PER     EQU     008H    ;PARITY ERROR
OVR     EQU     010H    ;DATA OVERRUN
BRK     EQU     020H    ;BREAK RECEIVED
CTS     EQU     040H    ;CLEAR TO SEND
DTD     EQU     080H    ;DIAL TONE DETECT
;
;--UART MODES (OUT MODEM0)
;
ASCII1  EQU     024H    ;7 DATA BITS, EVEN PARITY, 1 STOP BIT
ASCII2  EQU     004H    ;7 DATA BITS, ODD PARITY, 1 STOP BIT
BIN1    EQU     02CH    ;8 DATA BITS, EVEN PARITY, 1 STOP BIT
BIN2    EQU     00CH    ;8 DATA BITS, ODD PARITY, 1 STOP BIT
;
;--FILTER & CONTROL WORD SELECT CODES (OUT MODEM2)
;
MMSELO  EQU     001H    ;MODEM MODE CONTROL SELECT - ORIGINATE
MMSELA  EQU     031H    ;MODEM MODE CONTROL SELECT - ANSWER
DLSEL   EQU     002H    ;DIALER DIGIT CONTROL SELECT
BDSEL   EQU     004H    ;BAUD/DIAL RATE CONTROL SELECT
INTEN   EQU     008H    ;INTERRUPT ENABLE
;
;--MODEM MODES (OUT MODEM3)
```

```
;
MMORIG  EQU     0C0H    ;ORIGINATE: CH,DTR
MMLOW   EQU     043H    ;DTR,<=300,ESD,ELS
MMLREL  EQU     04BH    ;DTR,<=300,BRK-REL,ESD,ELS
MMHIGH  EQU     063H    ;DTR,>300,ESD,ELS
MMHREL  EQU     06BH    ;DTR,>300,BRK-REL,ESD,ELS
MMTERM  EQU     00EH    ;TERMINATE: BRK-REL,TX-BRK,ESD,ESS
MMBRK   EQU     004H    ;TX-BREAK
;
;--DIAL & BAUD RATE CONSTANTS (OUT MODEM3)
;
DRATE1  EQU     068H    ;DIAL RATE 10 PPS
DRATE2  EQU     034H    ;DIAL RATE 20 PPS
BAUD10  EQU     08EH    ;110 BAUD
BAUD15  EQU     068H    ;150 BAUD
BAUD30  EQU     034H    ;300 BAUD
BAUD45  EQU     022H    ;450 BAUD
BAUD50  EQU     01FH    ;500 BAUD
BAUD60  EQU     01AH    ;600 BAUD
;
;********************** CP/M EQUATES ******************************
;
; CP/M FUNCTIONS ARE EXECUTED BY PASSING THE FUNCTION NUMBER IN
; REGISTER C AND THE INFORMATION (ADDRESS OR DATA) IN REGISTERS
; D,E (REGISTER E IF SINGLE BYTE DATA).  THE RESULT (IF APPLICABLE)
; IS RETURNED IN REGISTER A.  IF DOUBLE BYTE RESULT, THE MSB IS IN
; REGISTER B.
;
;--FDOS FUNCTIONS
;
RESET   EQU     0       ;SYSTEM RESET
RDCON   EQU     1       ;READ CONSOLE CHARACTER
WRCON   EQU     2       ;WRITE CONSOLE CHARACTER
RDRDR   EQU     3       ;READ CHARACTER FROM READER DEVICE
WRPCH   EQU     4       ;WRITE CHARACTER TO PUNCH DEVICE
WRLST   EQU     5       ;WRITE CHARACTER TO LIST DEVICE
IOCHK   EQU     7       ;READ I/O CONFIGURATION BYTE
IOSET   EQU     8       ;WRITE I/O CONFIGURATION BYTE
WRCBUF  EQU     9       ;WRITE BUFFER CONTENTS TO CONSOLE
RDCBUF  EQU     10      ;READ BUFFER CONTENTS FROM CONSOLE
CONST   EQU     11      ;CONSOLE (DATA AVAILABLE) STATUS
;
UNLOAD  EQU     12      ;UNLOAD DISK R/W HEAD
DSKRES  EQU     13      ;DISK SYSTEM RESET
DSKSEL  EQU     14      ;DISK SELECT (0,1,2,3)
OPEN    EQU     15      ;OPEN DISK FILE (FCB ADDRESS)
CLOSE   EQU     16      ;CLOSE DISK FILE  "     "
SRCH    EQU     17      ;SEARCH FOR FILE  "     "
```

```
SRCHN   EQU     18      ;SEARCH FOR NEXT FILE
DELETE  EQU     19      ;DELETE DISK FILE (FCB ADDRESS)
RDDISK  EQU     20      ;READ NEXT RECORD   "   "
WRDISK  EQU     21      ;WRITE NEXT RECORD  "   "
CREATE  EQU     22      ;CREATE DISK FILE   "   "
RENAME  EQU     23      ;RENAME DISK FILE   "   "
LOGALL  EQU     24      ;IDENTIFY ALL ON-LINE DISKS
LOGSEL  EQU     25      ;IDENTIFY SELECTED DISK
SETDMA  EQU     26      ;SET DMA ADDRESS VALUE
;
;--CP/M ADDRESS CONSTANTS
;
BOOT    EQU     0       ;WARM START
FDOS    EQU     0005H   ;ENTRY TO FDOS
TFCB    EQU     005CH   ;TRANSIENT FILE CONTROL BLOCK
TBUFF   EQU     0080H   ;TRANSIENT BUFFER
;
;--FCB (FILE CONTROL BLOCK) CONSTANTS
;
FCB     EQU     TFCB
FCBDN   EQU     FCB+0   ;DISK NUMBER (0,1,2,3)
FCBFN   EQU     FCB+1   ;FILE NAME (8 BYTES)
FCBFT   EQU     FCB+9   ;FILE TYPE (3 BYTES)
FCBEX   EQU     FCB+12  ;EXTENT (0 TO 15)
FCBRC   EQU     FCB+15  ;REC COUNT (0 TO 128) - SIZE OF CURRENT EXTENT
FCBNR   EQU     FCB+32  ;NEXT RECORD POINTER (0 TO RC)
;
;****************** MISCELLANEOUS EQUATES **************************
;
;--ASCII CONTROL CODES
;
NHUL    EQU     000H    ;(^@) NULL (TIME FILL)
SOH     EQU     001H    ;(^A) START OF HEADING
STX     EQU     002H    ;(^B) START TEXT
ETX     EQU     003H    ;(^C) END TEXT
EOT     EQU     004H    ;(^D) END OF TRANSMISSION
ENQ     EQU     005H    ;(^E) ENQUIRY - "WHO ARE YOU"
ACK     EQU     006H    ;(^F) ACKNOWLEDGE - "YES"
BEL     EQU     007H    ;(^G) BELL - HUMAN ATTENTION REQUIRED
BS      EQU     008H    ;(^H) BACKSPACE
HT      EQU     009H    ;(^I) HORIZONTAL TABULATION
LF      EQU     00AH    ;(^J) LINE FEED
VT      EQU     00BH    ;(^K) VERTICAL TABULATION
FF      EQU     00CH    ;(^L) FORM FEED
CR      EQU     00DH    ;(^M) CARRIAGE RETURN
SO      EQU     00EH    ;(^N) SHIFT OUT - NONSTANDARD CODE FOLLOWS
SI      EQU     00FH    ;(^O) SHIFT IN - RETURN TO STANDARD CODE
;
DLE     EQU     010H    ;(^P) DATA LINK EXCAPE - CHANGE DATA COM CNTRL
```

```
DC1     EQU     011H    ;(^Q) DEVICE CONTROL 1
DC2     EQU     012H    ;(^R)   "       "    2
DC3     EQU     013H    ;(^S)   "       "    3
DC4     EQU     014H    ;(^T) DEVICE CONTROL 4
NAK     EQU     015H    ;(^U) NEGATIVE ACKNOWLEDGE - "NO"
SYN     EQU     016H    ;(^V) SYNCHRONOUS IDLE - ENABLES RESYNC
ETB     EQU     017H    ;(^W) END TRANSMISSION BLOCK
CAN     EQU     018H    ;(^X) CANCEL PREVIOUS DATA
EM      EQU     019H    ;(^Y) END MEDIUM
CLR     EQU     01AH    ;(^Z) CLEAR SCREEN ON LS-TERMINAL (ASCII: SUB)
ESC     EQU     01BH    ;(^[) ESCAPE - FOR CODE EXTENSION
FS      EQU     01CH    ;(^\) FILE SEPARATOR
GS      EQU     01DH    ;(^]) GROUP     "
RS      EQU     01EH    ;(^^) RECORD    "
US      EQU     01FH    ;(^_) UNIT      "
;
DEL     EQU     07FH    ;DELETE (RUBOUTS)
;
;--ZAPPLE I/O CONFIGURATION
;
ZIOCHK  EQU     0F015H
ZIOSET  EQU     0F018H
;
;################### PROGRAM STATEMENTS ##############################
;
        ORG     100H
        JMP     START
        DS      40      ;STACK AREA
STACK:  DS      2       ;STACK POINTER
MMSEL:  DS      A       ;MODEM MODE SELECT (ORIGINATE OR ANSWER)
MMODE:  DS      1       ;MODEM MODE
CHCNT:  DS      1       ;CHARACTER POSITION COUNT
BUFPTR: DS      1       ;DATA BUFFER POINTER
HSHAKE: DS      1       ;HAND-SHAKE FLAG
AUTOD:  DS      2       ;AUTO-DIAL FLAG
ROTFLG: DS      1       ;ROTARY DIALING FLAG
ANSFLG: DB      1       ;ANSWER MODE FLAG
LFDFLG  DB      1       ;LINE FEED FLAG
BLANKS: DB      '               ' ;16 BLANKS
;
;--CONSOLE DATA BUFFER
CBUFF   DB      16      ;BUFFER SIZE
CBCNT   DS      1       ;COUNT OF CHARACTERS READ
CBTXT   DS      16      ;CONSOLE DATA
;
BDTBL:  ;BAUD RATE VALUE TABLE
        DB      BAUD10  ;110 BAUD
        DB      BAUD15  ;150 BAUD
        DB      BAUD30  ;300 BAUD
```

```
        DB      BAUD45  ;450 BAUD
        DB      BAUD50  ;500 BAUD
        DB      BAUD60  ;600 BAUD
BDVAL:  DS      1       ;SELECTED BAUD RATE VALUE
;
;************************ PHONE NUMBERS ******************************
;
ADRTBL: ;PHONE NUMBER ADDRESS TABLE
        DW      PHONEH
        DW      PHONEM
        DW      PHONEZ
        DW      PHONEW
;
PHONEM: ;UNIV. OF MD.
        DB      '4544321$'
        DB      '4545201$'
        DB      '4545401$'
        DB      '4545405$'
        DB      GS      ;GROUP SEPERATOR
PHONEH: DB      '7765253$';HOME PHONE
PHONEZ: DB      '8441212$';TIME SERVICE
PHONEW: DB      '9361212$';WEATHER SERVICE
;
;************************ MESSAGES **********************************
;
MSG1    DB      CLR,'             88-MODEM DRIVER ROUTINE'
        DB      CR,LF,LF,LF
        DB      'TO INITIALIZE THE SYSTEM, TYPE ONE OF THE FOLLOWING '
        DB      'CHARACTERS:',CR,LF,LF
        DB      'H (HAND-SHAKE INIT.), B (BAUD-RATE SEL.), '
        DB      'CR (DEFAULT TO 300 BAUD)',CR,LF,LF,LF,BEL,'$'
MSG1A   DB      'SPECIFY THE MODE BY TYPING THE APPROPRIATE CHARACTER '
        DB      CR,LF,LF,'A --> ANSWER-MODE    O --> ORIGINATE-MODE'
        DB      CR,LF,LF,'$'
MSG1B   DB      CLR,'         ORIGINATE MODE -- WAITING FOR DIALTONE'
        DB      CR,LF,LF,'$'
MSG2    DB      BEL,'DIAL TONE RECEIVED:  COMMENSE DIALING - '
        DB      'TERMINATED BY A CR',CR,LF,LF,'$'
MSG2A   DB      LF,'WAITING FOR CARRIER DETECT',CR,LF,'$'
MSG3    DB      LF,LF,'CONNECTION HAS BEEN BROKEN',CR,LF,'$'
MSG4    DB      CLR,BEL,'CONNECTION HAS BEEN ESTABLISHED, BEGIN '
        DB      'NORMAL TRANSMISSION',CR,LF,LF,LF,'$'
MSG5    DB      CLR,BEL,'               RETURN TO CP/M EXECUTION'
        DB      CR,LF,LF,LF,'$'
MSG6    DB      BEL,'UNABLE TO OPEN FILE FOR TRANSMISSION',CR,LF,'$'
MSG7    DB      BEL,'DISK READ ERROR',CR,LF,'$'
MSG8    DB      BEL,'UNABLE TO CREATE FILE DUE TO INSUFFICIENT'
        DB      ' DIRECTORY SPACE',CR,LF,'$'
MSG9    DB      BEL,'FILE TRANSMISSION COMPLETED',CR,LF,LF,LF,'$'
```

```
MSG10   DB      CR,LF,LF,LF,'READY TO RECEIVE FILE',CR,LF,LF,LF,'$'
MSG11   DB      BEL,'UNABLE TO CLOSE FILE',CR,LF,'$'
MSG12   DB      BEL,'ERROR IN EXTENDING FILE',CR,LF,'$'
MSG13   DB      BEL,'DISK IS FULL',CR,LF,'$'
MSG14   DB      BEL,'DISK DIRECTORY IS FULL',CR,LF,'$'
MSG15   DB      '  HAND-SHAKE FLAG IS SET',CR,LF,LF,'$'
MSG16   DB      '  BAUD-RATE SELECTION',CR,LF,LF
        DB      'SELECT A BAUD RATE BY ENTERING APPROPRIATE DIGIT '
        DB      'FROM THE FOLLOWING LIST:',CR,LF
        DB      '1 (110 BAUD), 2 (150 BAUD), 3 (300 BAUD)',CR,LF
        DB      '4 (450 BAUD), 5 (500 BAUD), 6 (600 BAUD)',CR,LF,LF,'$'
MSG17   DB      CLR,'                          ANSWER MODE',CR,LF,LF,'$'
MSG18   DB      'WAITING FOR RING-INDICATOR AND CARRIER-DETECT'
        DB      CR,LF,LF,'$'
MSG19   DB      CR,LF,LF,'                       HELLO THERE',CR,LF,LF
        DB      '          PLEASE BEGIN YOUR TRANSMISSION',CR,LF,LF,'$'
MSG20   DB      CR,LF,LF,'ENTER FILE NAME IN 2 PARTS AS FOLLOWS:',CR,LF
        DB      '<FILENAME> CR',CR,LF
        DB      '<FILETYPE> CR',CR,LF,LF,'>$'
MSG21   DB      CR,LF,LF,'SELECT THE CONSOLE DEVICE FROM THE '
        DB      'FOLLOWING LIST:',CR,LF
        DB      '0 (TTY), 1 (CRT), 2 (BATCH), 3 (MODEM)',CR,LF,LF,'$'
;
;************************ SUBROUTINES *******************************
;
;++++++++++++++++++++++ COMMON SUBROUTINES +++++++++++++++++++++++++
;
EXIT:   ;RETURN TO CP/M
        MVI     C,WRCBUF
        LXI     D,MSG5
        CALL    FDOS    ;TYPE EXIT MESSAGE
        XRA     A       ;CLEAR A & CARRY
        LHLD    STACK
        SPHL            ;RESTORE CP/M STACK POINTER
        RET
;
CSTAT:  ;READ CONSOLE STATUS
        MVI     C,CONST
        CALL    FDOS
        ORA     A
        RET
;
CRD$ST: ;CONSOLE READ - GO TO "STOP" IF "^D"
        MVI     C,RDCON
        CALL    FDOS
        CPI     EOT
        RNZ             ;"^D" ?
        POP     PSW     ;  YES, CORRECT STACK FOR NON-RETURN
        XRA     A
```

```
        STA     ROTFLG  ;RESET ROTARY FLAG
        JMP     STOP
;
MSTAT:  ;READ MODEM STATUS AND DISPLAY ON PORT OFFH
        IN      MODEM0
        CMA
        OUT     0FFH
        CMA
        RET
;
MTBE:   ;MODEM TX BUFFER EMPTY TEST
        CALL    MSTAT
        ANI     TBE
        JZ      MTBE
        RET
;
TCHAR:  ;TYPE A CHARACTER ON CONSOLE DEVICE
        MVI     C,WRCON
        MOV     E,A
        CALL    FDOS
        RET
;
DCHAR:  ;DELETE A CHARACTER ON THE CONSOLE
        MVI     A,BS
        CALL    TCHAR
        MVI     A,' '
        CALL    TCHAR
        MVI     A,BS
        CALL    TCHAR
        MVI     A,BEL
        CALL    TCHAR
        JMP     DIAL
;
CRLF:   MVI     A,CR
        CALL    TCHAR
        MVI     A,LF
        CALL    TCHAR
        RET
;
THEX:   ;TYPE THE PAIR OF HEX CHARACTERS IN REG A
        PUSH    PSW
        RAR
        RAR
        RAR
        RAR
        CALL    TNIB
        POP     PSW
TNIB:   ANI     0FH
        CPI     10
```

```
        JC      ISNUM
        ADI     7
ISNUM:  ADI     '0'
        JMP     TCHAR
;
DIALIT: ;DIAL THE DIGIT IN REG A
        ANI     00FH
        OUT     MODEM3  ;SEND DIGIT TO DIALER
DBUSY:  CALL    CSTAT   ;GET CONSOLE STATUS
        JZ      D2      ;DATA AVAILABLE?
        CALL    CRD$ST  ;  YES, TEST FOR "^D"
D2:     CALL    MSTAT
        ANI     DBY
        JNZ     DBUSY   ;BUSY?
        MVI     A,'-'   ;  NO
        CALL    TCHAR   ;PROMPT FOR NEXT DIGIT
        RET
;
UART:   ;SET UP UART CONFIGURATION
        MVI     A,ASCII1
        OUT     MODEM0  ;SELECT THE NO. OF DATA BITS & PARITY TYPE
        MVI     A,BDSEL
        OUT     MODEM2
        LDA     BDVAL
        OUT     MODEM3  ;SET THE BAUD-RATE VALUE
        RET
;
MDM:    ;SET UP MODEM CONFIGURATION
        LDA     MMSEL
        OUT     MODEM2
        LDA     MMODE
        OUT     MODEM3
        NOP!    NOP!    NOP
        NOP!    NOP!    NOP
        ANI     0F7H    ;RESET THE "BREAK-RLS." BIT
        OUT     MODEM3
        RET
;
;+++++++++++++++++ CHARACTER TX/RX SUBROUTINES ++++++++++++++++++++++
;
BREAK:  ;TRANSMIT A BREAK TO REMOTE DEVICE
        LDA     MMSEL
        OUT     MODEM2
        LDA     MMODE
        ORA     MMBRK
        OUT     MODEM3  ;TRANSMIT A BREAK
        MVI     A,5
        CALL    DELAY   ;INSERT 0.5 SEC DELAY
        LDA     MMODE
```

```
        ANI     0F3H
        OUT     MODEM3  ;RESET BREAK & BREAK.RLS BITS
        JMP     TRCHAR
;
DELAY:  ;INSERT DELAY IN 100MS INCREMENTS (A.REG = INCREMENT.VAL)
        MVI     B,100
DEL1:   MVI     C,182   ;1MS = 182*(5.5 MICRO-SEC)
DEL2:   DCR     C
        JNZ     DEL2
        DCR     B
        JNZ     DEL1
        DCR     A
        JNZ     DELAY
        RET
;
MOVE:   ;SOURCE.ADDR=H,L   DEST.ADDR=D,E   BYTE.COUNT=A
        ORA     A
        RZ              ;BYTE.COUNT=0 ?
        MOV     B,A     ;  NO
MLOOP:  MOV     A,M
        XCHG
        MOV     M,A
        XCHG
        INX     D
        INX     H
        DCR     B
        JNZ     MLOOP
        RET
;
FILE:   ;SET UP FCB (FILE.CONTROL.BLOCK) AS SPECIFIED BY OPERATOR
        MVI     A,11
        LXI     H,BLANKS
        LXI     D,FCBFN
        CALL    MOVE    ;BLANK FILL FCB <FILENAME> <FILETYPE>
        MVI     C,WRCBUF
        LXI     D,MSG20
        CALL    FDOS    ;REQUEST <FILENAME> & <FILETYPE> FROM OPERATOR
        MVI     C,RDCBUF
        LXI     D,CBUFF
        CALL    FDOS    ;GET <FILENAME> FROM CONSOLE
        LDA     CBCNT
        LXI     H,CBTXT
        LXI     D,FCBFN
        CALL    MOVE    ;WRITE FCB <FILENAME>
        CALL    CRLF
        MVI     A,'>'
        CALL    TCHAR   ;PROMPT OPERATOR FOR <FILETYPE>
        MVI     C,RDCBUF
        LXI     D,CBUFF
```

```
        CALL    FDOS    ;GET <FILETYPE> FROM CONSOLE
        LDA     CBCNT
        LXI     H,CBTXT
        LXI     D,FCBFT
        CALL    MOVE    ;WRITE FCB <FILETYPE>
        CALL    CRLF
        JMP     TRCHAR
;
ASSIGN: ;MAKE CONSOLE ASSIGNMENT (TEMPORARILY LIMITED TO TTY & CRT)
        MVI     C,WRCBUF
        LXI     D,MSG21
        CALL    FDOS    ;PROMPT FOR CONSOLE SELECTION
        MVI     C,RDCON
        CALL    FDOS    ;READ SELECTION
        PUSH    PSW
        CALL    CRLF
        POP     PSW
        ANI     01H     ;TEMPORARILY ALLOWS FOR TTY & CRT ONLY
        PUSH    PSW     ;SAVE THE SELECTION
        CALL    ZIOCHK
        ANI     0FCH    ;MASK OUT THE 2 LSBS'
        MOV     B,A
        POP     PSW     ;GET THE SELECTION
        ORA     B
        MOV     C,A
        CALL    ZIOSET
        JMP     TRCHAR
;
;+++++++++++++++++++++ FILE TX SUBROUTINES ++++++++++++++++++++++++
;
OPENF:  ;OPEN DISK FILE FOR READING
        MVI     C,OPEN
        LXI     D,FCB
        CALL    FDOS
        CPI     255
        JNZ     OPNOK   ;IS FILE IN DIRECTORY?
        MVI     C,WRCBUF ;  NO
        LXI     D,MSG6
        CALL    FDOS    ;TYPE ERROR MESSAGE
        POP     PSW     ;CORRECT STACK PTR FOR NON-RETURN
        JMP     TRCHAR  ;REVERT TO CHARACTER MODE
OPNOK:  XRA     A
        STA     FCBNR   ;NEXT-RECORD = 0
        RET
;
DISKR:  ;READ A DISK SECTOR (RECORD)
        MVI     C,RDDISK
        LXI     D,FCB
        CALL    FDOS
```

```
        CPI     0
        RZ              ;NORMAL READ?
        CPI     1       ;  NO
        JNZ     DRERR   ;END OF FILE?
        MVI     C,WRCBUF ;  YES
        LXI     D,MSG9
        CALL    FDOS    ;TYPE "END FILE" MSG
        JMP     DRERRX
DRERR:  MVI     C,WRCBUF
        LXI     D,MSG7
        CALL    FDOS    ;TYPE "READ ERR" MSG
DRERRX: POP     PSW     ;CORRECT STACK FOR NON-RETURN
        JMP     TRCHAR  ;REVERT TO CHARACTER MODE
;
SHOW:   ;SHOW CHAR TRANSMITTED OR RECEIVED
        CPI     LF
        JZ      TCHAR
        CPI     CR
        JZ      TCHAR
        CPI     ' '
        JC      SHEX    ;ASCII CONTROL CHARACTER?
        CPI     07FH    ;  NO
        JC      TCHAR   ;ASCII PRINTABLE CHARACTER?
SHEX:   ;SHOW HEX VALUE FOR NON-PRINTABLE CHARACTER
        PUSH    PSW     ;  NO
        MVI     A,'('
        CALL    TCHAR
        POP     PSW
        CALL    THEX
        MVI     A,')'
        CALL    TCHAR
        RET
;
;++++++++++++++++++++++ FILE RX SUBROUTINES ++++++++++++++++++++++++
;
MAKEF:  ;CREATE A FILE FOR SUBSEQUENT WRITING
        MVI     C,CREATE
        LXI     D,FCB
        CALL    FDOS
        CPI     255
        JNZ     MAKEOK  ;IS SUFFICIENT DIRECTORY SPACE AVAILABLE?
        MVI     C,WRCBUF ;  NO
        LXI     D,MSG8
        CALL    FDOS    ;TYPE ERROR MSG.
        POP     PSW     ;CORRECT STACK FOR NON-RETURN
        JMP     TRCHAR  ;REVERT TO CHARACTER MODE
MAKEOK: XRA     A
        STA     FCBNR   ;NEXT-RECORD = 0
        RET
```

```
;
ST$BUF: ;STORE CONTENTS OF REG A IN DISK BUFFER
        PUSH    PSW
        LDA     BUFPTR
        MOV     E,A     ;D,E = DATA BYTE ADDRESS
        MVI     D,0
        INR     A
        STA     BUFPTR  ;INCREMENTED BUFFER POINTER
;--STORE A BYTE A ADDRESS: TBUFF+(D,E)
        LXI     H,TBUFF
        DAD     D
        POP     PSW
        MOV     M,A
;--TEST FOR BUFFER FULL
        LDA     BUFPTR
        CPI     080H
        RNZ             ;IS BUFFER FULL?
        POP     PSW     ;YES, CORRECT STACK FOR NON-RETURN
        JMP     RFULL
;
DISKW:  ;WRITE A DISK SECTOR (RECORD)
        MVI     C,WRDISK
        LXI     D,FCB
        CALL    FDOS
        CPI     0
        RZ              ;NORMAL WRITE?
        CPI     1       ;  NO
        JNZ     DR1     ;ERROR IN EXTENDING FILE?
        MVI     C,WRCBUF ;  YES
        LXI     D,MSG12
        CALL    FDOS    ;TYPE "EXTEND ERROR" MSG
        JMP     DWERRX
DR1:    CPI     2
        JNZ     DR2     ;IS DISK FULL?
        MVI     C,WRCBUF ;  YES
        LXI     D,MSG13
        CALL    FDOS    ;TYPE "DISK FULL" MSG
        JMP     DWERRX
DR2:    MVI     C,WRCBUF
        LXI     D,MSG14
        CALL    FDOS    ;TYPE "DIRECTORY FULL" MSG
DWERRX: POP     PSW     ;CORRECT STACK FOR NON-RETURN
        JMP     TRCHAR  ;REVERT TO CHARACTER MODE
;
CLOSEF: ;CLOSE DISK FILE (WRITING COMPLETED)
        MVI     C,CLOSE
        LXI     D,FCB
        CALL    FDOS
        CPI     255
```

```
        RNZ             ;SUCESSFUL CLOSE?
        MVI     C,WRCBUF ;  NO
        LXI     D,MSG11
        CALL    FDOS    ;TYPE ERROR MESSAGE
        RET
;
;********************** MAIN PROGRAM ***************************
;
START:  ;INITIALIZE OUR STACK
        LXI     H,0
        DAD     SP      ;HL=STACK FROM CP/M
        SHLD    STACK   ;SAVE CP/M STACK POINTER
        LXI     SP,STACK ;LOAD OUR STACK PIONTER
RESTRT: XRA     A
        STA     HSHAKE  ;RESET HAND-SHAKE FLAG
        STA     AUTOD   ;RESET AUTO-DIAL FLAG
        STA     ROTFLG  ;RESET ROTARY DIALING FLAG
        STA     ANSFLG  ;RESET ANSWER MODE FLAG
        STA     LFDFLG  ;RESET LINE FEED FLAG
        MVI     A,BAUD30
        STA     BDVAL   ;DEFAULT BAUD RATE VALUE = 300
        MVI     A,MMLREL
        STA     MMODE   ;DEFAULT MODEM MODE = MMLREL
        MVI     C,WRCBUF
        LXI     D,MSG1
        CALL    FDOS    ;TYPE SIGN-ON MESSAGE
;--WAIT FOR CONSOLE INPUT
        MVI     C,RDCON
        CALL    FDOS    ;READ CONSOLE DATA
        CPI     EOT
        JZ      EXIT    ;"^D"?
        ANI     05FH    ;  NO, SHIFT LC ALPLH TO UC
        CPI     'H'
        JNZ     BAUD    ;HAND-SHAKE REQUEST?
        MVI     A,0FFH  ;  YES
        STA     HSHAKE  ;SET THE HAND-SHAKE FLAG TO ON
        MVI     C,WRCBUF
        LXI     D,MSG15
        CALL    FDOS    ;TYPE "HAND-SHAKE" MSG.
        JMP     DO
;--BAUD RATE MODIFICATION
BAUD:   CPI     'B'
        JNZ     DO      ;BAUD-RATE CHANGE REQUEST?
        MVI     C,WRCBUF;  YES
        LXI     D,MSG16
        CALL    FDOS    ;PROMPT FOR BAUD SELECTION
BA1:    CALL    CSTAT
        JZ      BA1     ;HAS OPERATOR RESPONDED?
        MVI     C,RDCON ;  YES, GET THE CHARACTER
```

```
        CALL    FDOS
        PUSH    PSW
        CPI     '4'
        JC      BA2     ;BAUD-RATE > 300 ?
        MVI     A,MMHREL        ;  YES, CHANGE MODEM MODE ACCORDINGLY
        STA     MMODE
BA2:    CALL    CRLF
        POP     PSW
        ANI     07H
        DCR     A
        MOV     E,A
        MVI     D,00H   ;D,E = TBL INDEX
        LXI     H,BDTBL
        DAD     D
        MOV     A,M
        STA     BDVAL   ;NEW TRANSMISSION BAUD-RATE VALUE
;--DETERMINE MODEM MODE (ORIGINATE/ANSWER)
DO:     MVI     C,WRCBUF
        LXI     D,MSG1A
        CALL    FDOS    ;TYPE "MODE" MESSAGE
        MVI     C,RDCON
        CALL    FDOS    ;READ CONSOLE DATA
        CPI     EOT
        JZ      RESTRT  ;"^D"?
        ANI     05FH    ;  NO, SHIFT LC ALPHA TO UC
        CPI     'O'
        JZ      ORIG    ;ORIGINATE-MODE SELECTED?
        CPI     'A'     ;  NO
        JNZ     DO      ;ANSWER-MODE SELEDTED?
        MVI     A,0FFH  ;  YES
        STA     ANSFLG
        JMP     ANSWER
;
;++++++++++++++++++++++ ORIGINATE MODE ++++++++++++++++++++++++++++
;
;--SET UP DIALER BAUD RATE
ORIG:   MVI     A,BDSEL ;  NO
        OUT     MODEM2
        MVI     A,DRATE2
        OUT     MODEM3
;--GO OFF-HOOK & SET BREAK RELEASE
        MVI     A,MMSELO
        OUT     MODEM2
        MVI     A,MMORIG
        OUT     MODEM3
        MVI     C,WRCBUF
        LXI     D,MSG1B
        CALL    FDOS    ;TYPE "WAITING" MESSAGE
;--WAIT FOR DIAL TONE OR "^D"
```

```
DTONE:  CALL    CSTAT   ;GET CONSOLE STATUS
        JZ      D1      ;CONSOLE DATA AVAILABLE?
        CALL    CRD$ST  ;  YES, TEST FOR "^D"
D1:     CALL    MSTAT   ;GET MODEM STATUS
        ANI     DTD
        JNZ     DTONE   ;DIAL TONE PRESENT?
        MVI     C,WRCBUF ;  YES, INFORM OPERATOR
        LXI     D,MSG2
        CALL    FDOS
;--BEGIN DIALING SEQUENCE
        MVI     A,DLSEL
        OUT     MODEM2
        LDA     ROTFLG
        ORA     A
        JZ      DIAL    ;ROTARY DIALING IN PROGRESS?
        POP     PSW     ;  YES, GET ROTARY CHARACTER
        JMP     AUTODL
DIAL:   CALL    CRD$ST  ;CONSOLE READ
        CPI     CR
        JZ      UARTO   ;CARRIAGE RETURN?
;--SCREEN OUT NON-DECIMAL "DIALED" CHARACTERS
        CPI     '0'     ;  NO
        JC      DCHAR   ;CONTROL CHAR. OR SYMBOL?
        CPI     ':'     ;  NO
        JC      DIGIT   ;NON-DECIMAL CHARACTER?
        JMP     AUTODL  ;  YES, PERFORM AUTO-DIALING
DIGIT:  CALL    DIALIT  ;DIAL A DIGIT
        JMP     DIAL
;--AUTOMATIC DIALING ROUTINE
AUTODL: LXI     H,0FFFFH
        SHLD    AUTOD   ;SET AUTO-DIAL FLAG
        PUSH    PSW     ;SAVE CHARACTER FOR ROTARY DIALING TEST
        ANI     03H
        RLC
        MOV     E,A
        MVI     D,0
        LXI     H,ADRTBL
        DAD     D       ;H,L=INDIRECT ADDRESS OF PHONE NUMBER
        MOV     E,M
        INX     H
        MOV     D,M
        XCHG            ;H,L=ADDRESS OF PHONE NUMBER
AU1:    MOV     A,M
        CPI     GS
        JZ      ROTEND  ;END OF ROTARY GROUP?
        CPI     '$'     ;  NO
        JZ      UARTO-1 ;DIALING COMPLETE?
        PUSH    H       ;  NO
        PUSH    PSW
```

```
        CALL    TCHAR   ;TYPE DIGIT TO BE DIALED
        POP     PSW
        CALL    DIALIT
        POP     H
        INX     H
        JMP     AU1
;--SET UP UART CONFIGURATION
        PUSH    H       ;SAVE ADDRESS POINTER FOR ROTARY DIALING
UARTO:  MVI     C,WRCON
        MVI     E,LF
        CALL    FDOS    ;SEND "LINE-FEED" TO CONSOLE
        CALL    UART    ;ESTABLISH UART CONFIGURATION
;--WAIT FOR CTS (CARRIER DETECT)
        MVI     C,WRCBUF
        LXI     D,MSG2A
        CALL    FDOS    ;TYPE "WAITING" MESSAGE
        LXI     B,0FFFFH ;INITIALIZE TIME-OUT
        PUSH    B
CTS1:   POP     B
        DCX     B
        MOV     A,B
        ORA     C
        JZ      ROTARY  ;TIME-OUT EXPIRED?
        PUSH    B       ;  NO
        CALL    CSTAT   ;GET CONSOLE STATUS
        JZ      C1      ;CONSOLE DATA AVAILABLE?
        CALL    CRD$ST  ;  YES, TEST FOR "^D"
C1:     CALL    MSTAT
        ANI     CTS
        JZ      CTS1    ;CLEAR TO SEND?
        JMP     MDMO    ;  YES, SET UP MODEM CONFIGURATION
;--ROTARY DIALING SEQUENCE
ROTARY: LDA     AUTOD
        ORA     A
        JZ      STOP    ;AUTO-DIAL MODE?
        POP     H       ;  YES, GET PH# ADDRESS
        INX     H
        POP     PSW     ;GET CHARACTER WHICH INITIATED AUTO-DIAL
        CPI     'M'
        JNZ     STOP    ;ROTARY CHARACTER?
        PUSH    PSW     ;  YES, SAVE FOR NEXT ITERATION
        MVI     A,0FFH
        STA     ROTFLG  ;SET ROTARY DIALING FLAG
        SHLD    ADRTBL+2;STORE INCREMENTED POINTER
        JMP     STOP    ;TRY THE NEXT PHONE NUMBER
ROTEND: LXI     H,PHONEM
        SHLD    ADRTBL+2;RESTORE ORIGINAL POINTER
        XRA     A
        STA     ROTFLG  ;RESET ROTARY DIALING FLAG
```

```
        POP     PSW     ;RESTORE STACK TO NORMAL
        JMP     STOP
;--SET UP MODEM CONFIGURATION
MDMO:   MVI     A,MMSELO
        STA     MMSEL
        CALL    MDM
;--PROMPT OPERATOR TO BEGIN COMMUNICATIONS
        MVI     C,WRCBUF
        LXI     D,MSG4
        CALL    FDOS    ;TYPE START TX MESSAGE
        XRA     A
        STA     ROTFLG  ;RESET ROTARY DIALING FLAG
        JMP     TRCHAR
;
;++++++++++++++++++++++ ANSWER MODE ++++++++++++++++++++++++++++++
;
ANSWER: MVI     C,WRCBUF
        LXI     D,MSG17
        CALL    FDOS    ;TYPE "ANSWER MODE" MSG
        MVI     A,0FFH
        STA     LFDFLG  ;SET LINE.FEED FLAG
        CALL    UART    ;SET UP UART CONFIGURATION
        MVI     A,MMSELA
        STA     MMSEL
        CALL    MDM     ;SET UP MODEM CONFIGURATION
        MVI     C,WRCBUF
        LXI     D,MSG18
        CALL    FDOS    ;TYPE "WAITING" MESSAGE
;--WAIT FOR CTS (RING INDICATOR & CARRIER DETECT)
ANO:    CALL    CSTAT
        JZ      AN1     ;CONSOLE DATA AVAILABLE?
        MVI     C,RDCON ;  YES
        CALL    FDOS    ;READ CONSOLE DATA
        CPI     EOT
        JNZ     AN1     ;"^D"?
        XRA     A       ;  YES
        STA     ANSFLG  ;RESET ANSWER MODE FLAG
        JMP     STOP
AN1:    IN      MODEMO  ;  NO
        ANI     CTS
        JZ      ANO     ;HAS CONNECTION BEEN ESTABLISHED?
        MVI     C,WRCBUF;  YES, INFORM CONSOLE OPERATOR
        LXI     D,MSG4
        CALL    FDOS
;--TRANSMIT SIGN-ON MESSAGE TO REMOTE DEVICE
        LXI     H,MSG19 ;H,L = MESSAGE ADDRESS
SIGNON: CALL    MSTAT
        ANI     TBE
        JZ      SIGNON  ;TX BUFFER EMPTY?
```

```
        MOV     A,M     ;  YES, GET NEXT CHARACTER
        CPI     '$'
        JZ      TRCHAR  ;MESSAGE COMPLETE?
        OUT     MODEM1  ;  NO, TRANSMIT THE CHARACTER
        INX     H
        JMP     SIGNON
;
;+++++++++++++++++ CHARACTER TX/RX ROUTINE ++++++++++++++++++++++++++
;
;--TEST FOR LOSS OF "CTS"
TRCHAR: IN      MODEM0
        ANI     CTS
        JZ      STOP    ;LOSS OF CTS?
;--TEST FOR "LINE CHARACTER" AVAILABILITY
        IN      MODEM0  ;  NO
        ANI     RDA
        JZ      NOCHAR  ;RECEIVE DATA AVAILABLE?
        IN      MODEM1  ;  YES, READ DATA
        PUSH    PSW
        CALL    TCHAR   ;TYPE RECEIVED CHARACTER
        LDA     LFDFLG
        ORA     A
        JZ      TR1     ;LINE FEED FLAG SET?
        POP     PSW     ;  YES
        CPI     CR
        JNZ     TR1+1   ;CHARRIAGE RETURN?
TRLF:   MVI     A,LF    ;  YES, SEND A LINE FEED
        CALL    TCHAR
TR0:    IN      MODEM0
        ANI     TBE
        JZ      TR0     ;TX BUFFER EMPTY?
        MVI     A,LF    ;  YES
        OUT     MODEM1  ;TRANSMIT LINE FEED TO LINE
        JMP     NOCHAR
TR1:    POP     PSW
        CPI     DC4
        JZ      TXFILE  ;"^T"?
        CPI     DC2     ;  NO
        JZ      RXFILE  ;"^R"?
        CPI     EOT     ;  NO
        JNZ     NOCHAR  ;"^D"?
        XRA     A       ;  YES
        STA     ANSFLG  ;RESET THE ANSER.MODE FLAG
        JMP     STOP    ;TERMINATE ANSWER MODE
;--TEST FOR "LOCAL CHARACTER" AVAILABILITY
NOCHAR: CALL    CSTAT   ;GET CONSOLE STATUS
        JZ      TRCHAR  ;CONSOLE DATA AVAILABLE?
        CALL    CRD$ST  ;  YES, READ DATA & TEST FOR "^D"
        CPI     DC4
```

```
        JZ      TXFILE  ;"^T"?
        CPI     DC2     ;  NO
        JZ      RXFILE  ;"^R"?
        CPI     SOH
        JZ      ASSIGN  ;IF "^A" THEN CHANGE CONSOLE ASSIGNMENT
        CPI     STX
        JZ      BREAK   ;IF "^B" THEN TRANSMIT A "BREAK"
        CPI     ACK
        JZ      FILE    ;IF "^F" THEN SET UP NEW FILE NAME
        OUT     MODEM1  ;TRANSMIT DATA TO "LINE"
        CPI     CR
        JNZ     TRCHAR  ;CARRIAGE RETURN?
        LDA     LFDFLG  ;  YES
        ORA     A
        JNZ     TRLF    ;LINE FEED FLAG SET?
        JMP     TRCHAR  ;  NO
;
;++++++++++++++++++++ FILE TX ROUTINE ++++++++++++++++++++++++++
;
TXFILE: CALL    OPENF   ;OPEN DISK FILE FOR READING
        MVI     A,80H
        STA     BUFPTR  ;INITIALIZE THE BUFFER POINTER
        XRA     A
        STA     CHCNT   ;INITIALIZE CHARACTER POSITION COUNT
;--START OF BYTE TRANSMIT LOOP
TXBYTE: CALL    CSTAT   ;GET CONSOLE STATUS
        JZ      TF0     ;DATA AVAILABLE?
        MVI     C,RDCON ;  YES
        CALL    FDOS
        CPI     EOT
        JZ      TRCHAR  ;"^D" ?
TF0:    IN      MODEM0  ;  NO
        ANI     RDA
        JZ      TF1A    ;RX DATA AVAILABLE?
        IN      MODEM1  ;  YES
        CPI     EOT
        JZ      TRCHAR  ;"^D"?
TF1A:   LDA     BUFPTR  ;  NO
        CPI     80H
        JNZ     TF1     ;BUFFER EMPTY?
;--READ A SECTOR & EXIT IF END OF FILE
        CALL    DISKR   ;  YES
        XRA     A
TF1:    MOV     E,A     ;D,E = DATA BYTE ADDRESS
        MVI     D,0
        INR     A
        STA     BUFPTR  ;INCREMENTED BUFFER POINTER
;--READ A BYTE AT ADDRESS: TBUFF+(D,E)
        LXI     H,TBUFF
```

```
        DAD     D
        MOV     A,M
        PUSH    PSW
;--WAIT FOR UART TX BUFFER EMPTY
TF2:    CALL    MSTAT
        ANI     TBE
        JZ      TF2     ;UART TX BUFFER EMPTY?
        POP     PSW     ;  YES
        CPI     ETB
        JZ      TFEND   ;(^W) END TRANSMISSION BLOCK?
        CPI     HT      ;  NO
        JNZ     TF3     ;HORIZONTAL TAB?
;--TRANSLATE HT INTO APPROPRIATE NUMBER OF SPACES
        LDA     CHCNT   ;  YES
SPACES: MOV     B,A
        CALL    MTBE    ;INSURE MODEM TX BUFFER EMPTY
        MVI     A,' '
        OUT     MODEM1
        PUSH    B
        CALL    TCHAR
        POP     B
        INR     B
        MOV     A,B
        ANI     07H     ;MOD 8 COUNT VALUE
        JNZ     SPACES  ;FINAL SPACE SENT?
        STA     CHCNT
        JMP     TXBYTE
;--SEND DATA BYTE & UPDATE CHCNT (CHARACTER POSITION COUNT)
TF3:    PUSH    PSW
        CALL    SHOW    ;DISPLAY BYTE ON CONSOLE
        POP     PSW
        OUT     MODEM1
        CPI     LF
        JZ      TFEOL   ;"LF" SIGNALS END OF TEXT LINE
        LDA     CHCNT
        INR     A
        STA     CHCNT   ;INCREMENTED CHAR POSITION COUNT
        JMP     TXBYTE
;--PERFORM END OF LINE PROCESSING
TFEOL:  XRA     A
        STA     CHCNT   ;RESET CHAR POSITION COUNT
        LDA     HSHAKE
        ORA     A
        JNZ     TF4     ;HAND-SHAKE REQUIRED?
        MVI     A,5     ;  NO, GENERATE 0.5 SEC DELAY
        CALL    DELAY
        JMP     TXBYTE  ;BEGIN NEXT LINE
;--PERFORM END OF TEXT LINE "HANDSHAKING"
TF4:    CALL    CSTAT
```

```
        JZ      TF4A    ;CONSOLE CHAR RECEIVED?
        JMP     TFEND   ;  YES, TERMINATE FILE TX
TF4A:   CALL    MSTAT   ;
        ANI     RDA
        JZ      TF4     ;RECEIVE DATA AVAILABLE?
        IN      MODEM1  ;  YES
        CPI     LF
        JNZ     TF4     ;"LINE FEED" RECEIVED FROM LINE?
        MVI     A,1     ;  YES, INSERT 100MS DELAY
        CALL    DELAY
        JMP     TXBYTE  ;BEGIN NEXT LINE
TFEND:  CALL    CRLF
        JMP     TRCHAR  ;REVERT TO CHARACTER MODE
;
;++++++++++++++++++++ FILE RX ROUTINE ++++++++++++++++++++++++++
;
RXFILE: MVI     C,DELETE
        LXI     D,FCB
        CALL    FDOS    ;DELETE NAMED FILE (IF IT EXISTS)
        CALL    MAKEF   ;CREATE FILE FOR RX DATA
        XRA     A
        STA     BUFPTR  ;INITIALIZE THE BUFFER POINTER
        MVI     C,WRCBUF
        LXI     D,MSG10
        CALL    FDOS    ;TYPE "READY" MSG
;--RECEIVE A CHARACTER FROM THE REMOTE DEVICE
RCV:    IN      MODEM0
        ANI     RDA
        JNZ     GOTONE  ;RECEIVED DATA AVAILABLE?
;--SUPPORT CHARACTER TX FROM CONSOLE
        CALL    CSTAT   ;  YES, GET CONSOLE STATUS
        JZ      RCV     ;DATA AVAILABLE?
        MVI     C,RDCON ;  YES
        CALL    FDOS
        CPI     EOT
        JZ      RFEND   ;"^D" ?
        OUT     MODEM1  ;  NO, TRANSMIT CHARACTER
        JMP     RCV
;--READ CHARACTER FROM EXTERNAL DEVICE
GOTONE: IN      MODEM1
        CPI     ETB
        JZ      RF0     ;"^W"?
        CPI     EOT     ;  NO
        JZ      RFEND   ;"^D"?
        CPI     CR      ;  NO
        JZ      RFCR    ;CARRIAGE RETURN?
        CPI     DEL     ;  NO
        JZ      RCV     ;DO NOT RECORD RUBOUTS
        CPI     ' '
```

```
        JC      RCV     ;ASCII CONTROL CHARACTER?
        PUSH    PSW     ;  NO
        CALL    TCHAR   ;DISPLAY CHARACTER ON CONSOLE
        POP     PSW
RF0:    CALL    ST$BUF  ;STORE CHARACTER IN DISK BUFFER
        JMP     RCV
;--CARRIAGE RETURN PROCESSING
RFCR:   PUSH    PSW
        CALL    TCHAR   ;TYPE "CR"
        POP     PSW
        CALL    ST$BUF  ;PUT IT IN BUFFER
        MVI     A,LF
        PUSH    PSW
        CALL    TCHAR   ;TYPE "LF"
        POP     PSW
        CALL    ST$BUF  ;PUT IT IN BUFFER
        LDA     LFDFLG
        ORA     A
        JZ      RCV     ;LINE.FEED FLAG SET?
RF1:    IN      MODEM0  ;  YES
        ANI     TBE
        JZ      RF1     ;TX BUFFER EMPTY?
        MVI     A,LF    ;  YES
        OUT     MODEM1  ;SEND LINE.FEED TO REMOTE DEVICE
        JMP     RCV
;--BUFFER IS FULL - WRITE A DISK SECTOR
RFULL:  CALL    DISKW   ;  YES
        XRA     A
        STA     BUFPTR
        JMP     RCV
;--WRITE FINAL SECTOR & CLOSE FILE
RFEND:  LDA     BUFPTR
        MOV     E,A
        MVI     D,0
        LXI     H,TBUFF
        DAD     D
        MVI     A,01AH  ;"^Z" INDICATES END OF FILE
        MOV     M,A
        CALL    DISKW   ;WRITE THE LAST RECORD
        CALL    CLOSEF
        JMP     TRCHAR  ;REVERT TO CHARACTER MODE
;
;+++++++++++++++++ TERMINATE COMMUNICATIONS ++++++++++++++++++++++
;
;--SEND A BREAK, HANG UP THE PHONE, INFORM THE OPERATOR
STOP:   MVI     A,MMSELO
        OUT     MODEM2
        MVI     A,MMTERM
        OUT     MODEM3
```

```
        MVI     C,WRCBUF
        LXI     D,MSG3
        CALL    FDOS    ;TYPE TERMINATION MESSAGE
;--CONTINUE TO LOOP TILL LOSS OF CTS (CARRIER DETECT)
CTS2:   CALL    MSTAT
        ANI     CTS
        JNZ     CTS2    ;HAS CTS BEEN RESET?
        MVI     A,30    ;   YES
        CALL    DELAY   ;3 SECOND DELAY
        LDA     ROTFLG
        ORA     A
        JNZ     ORIG    ;ROTARY DIALING IN PROGRESS?
        LDA     ANSFLG  ;  NO
        ORA     A
        JNZ     ANSWER  ;ANSWER MODE?
        JMP     RESTRT  ;  NO
        END     START

OG>
```

Glossary

application program—a program for a particular end use such as a word processor, general ledger, or mailing list processor.

ASCII (American National Standard Code for Information Interchange)—a standard code of seven or eight bits used for information exchange between computer systems.

ASCII files—data files consisting entirely of ASCII characters, such as a letter file for a word processing program.

assemble—to translate a program from an assembly language to a computer machine language.

assembler—a computer program that converts a program in a symbolic machine language to an executable form in machine language. The CP/M assembler converts a source program to a HEX file form, and requires the LOAD program to convert this to an executable form.

assembly-level program—a program in which the instructions are usually one-to-one correspondence with computer instructions. Such a program is highly specific to a particular type of microprocessor (as 8080 or 6502) and requires extensive modifications to use on another type of system.

bank select—a method of switching memory modules active or inactive with I/O instructions from the CPU. Used in multi-user systems to provide each user with a full memory bank.

baud rate—the transmission rate of information between devices in bits per second. A fast IBM Selectric runs at roughly 150 baud, while a low speed modem used in microprocessor systems is 300 baud. Video terminals used as a console are normally 4800 or 9600 baud.

BDOS (Basic Disk Operating System)—the CP/M routines for the management of disk read and write operations. The routines are hardware independent.

bias—the distance in memory of the new CCP to be generated from the CCP of the distributed CP/M.

BIOS (Basic I/O System)—the module of CP/M that contains all the input and output routines for all the peripherals. This is the only part of CP/M that is hardware dependent.

BIOS software driver—an I/O routine in the BIOS module, such as the printer driver.

bootstrap—a short computer program used to load a longer program.

byte—a group of eight bits that represents a single character.

bus—one or more conductors for the transmission of signals or power.

CCP (Console Command Processor)—the part of CP/M that interprets keyboard input.

COBOL (Common Business-Oriented Language)—a high-level language designed specifically for business applications.

COM files—CP/M program files on the disk that are machine-level language and can be called as programs.

compiler—a high-level program that converts another program from a problem-oriented language to a machine-oriented language.

Data Base Management System (DBMS)—a programming system that permits you to organize, manipulate, and report data easily in a user-oriented form.

destination disk—the disk being created (i.e., receiving data).

device assignments–the relationship of the logical devices to the physical devices used in a computer system.

Direct Memory Access (DMA)–any method of moving data directly from a peripheral to memory (or vice versa) without the continued use of the CPU. The CPU is used to set up the transfer, then the transfer continues while the bus is not being used by the CPU.

directory–an index used by a control program to locate one or more blocks of data on a disk.

directory information–information concerning file locations, lengths, and sizes. Normally stored on track 3 of a diskette in a CP/M system.

disk controller–the hardware used to manage the disk operation.

diskette–a thin magnetic disk in a semi-rigid protective jacket.

dynamic file allocation–managing disk space in such a way so that if files are deleted or altered, disk space is automatically allocated and recovered without having to repack the information on the disk.

extent–the maximum amount of disk space that can be allocated by a single CP/M directory entry. The CP/M extent size is 16K. If a file requires more space than this, CP/M automatically creates additional directory entries. Most directory listings (such as DIR) will show all file extents as a single directory entry.

FDOS–the combined BDOS/BIOS modiles of CP/M.

file–a collection of related records treated as a single item.

file transfer–the act of moving a collection of related items identified as a unit from one place (such as the disk) to another (such as a second disk).

firmware–computer hardware with information content, such as a PROM with a program stored within its memory.

FORTRAN (FORmula TRANslation)–a high-level programming language used primarily for scientific applications.

group—the smallest unit of assignable disk space in a CP/M directory. An extent can consist of up to sixteen groups. The amount of storage space in a group, however, can vary with different types of CP/M systems.

hardware—computer resources with physical visibility (contrasts with firmware and software).

Indexed Sequential Access (or ISAM)—a method of storing information in a sequential file, and maintaining a keyed access to the file. Allows quick access of any desired record.

interpreter—a high-level language that resides in computer memory along with the application program and translates the program to a machine-level language on a statement-by-statement basis as the application program is executed.

invariant portion—that part of the CP/M that does not vary with hardware. The BDOS and CCP modules.

I/O—pertaining to a device or channel that may be involved in an input process and, at another time, in an output process.

IOBYTE—location 0003H in a CP/M system. Controls the mapping of logical to physical drivers.

kernel—that part of the operating system that always remains in the computer memory.

linked directory—the directory concept of storing pointers in the directory records that link common directory entries for a given file. With time, the disk can become cluttered and must be repacked, releasing deleted space. Some linked systems dynamically repack the disk.

LOAD program—a program that converts CP/M files stored in HEX format to a machine-level COM program file.

logical device—a peripheral referenced by a computer program. Some CP/M systems can use STAT to control the relationship of logical to physical devices (see physical device).

modem—a device that transforms computer data into a form that can be sent over a telephone line or radio link. A modem also receives such information and reconverts it into computer data.

monitor—software or firmware that observes, supervises, controls, and verifies the operation of a system.

multi-user operating system—an operating system that can be used by more than one user at a time.

negative offset—in generating a new CP/M system, this is the distance of the BIOS image of the new system to be generated from the BIOS image of the old system.

null directory entry—a directory entry for an empty file (contains no information).

object file—the machine-level program produced as the output of an assembly or compile operation.

offset—(see negative offset.)

operating system—a group of programs that organizes a collection of hardware devices into an integrated working system.

page—a 256-byte memory space (in most microcomputer systems).

parameters—information passed by the user to a program to control execution.

peripheral—any part of the computer system external to the central processor that provides additional features: a printer, modem, or terminal.

physical device—a peripheral and its associated BIOS driver in CP/M. A "real" device as contrasted to the logical device seen by the program. CP/M manages the relationship between logical and physical devices.

PIP program—the CP/M Peripheral Interchange Program. Used to load, print, punch, copy, or combine files.

PL/1—a high-level language compiler originally developed in 1964 by IBM and its two user groups. It was an attempt to provide a common computer language for both the scientific and business communities. A CP/M version is available.

PL/M (Programming Language for Microcomputers)—a high-level structured language for microcomputers that contains a full set of PL/1 commands and elements of ALGOL. It is a refinement of the XPL compiler writing language.

primitive—the basic or lowest unit of use in a programming language. In the CP/M operating system, it is used to refer to the BDOS function calls available.

preprocessor—a high-level language used to compile a source code to a specific form that can be read by another compiler.

processor—the functional part of the computer system that reads and interprets instructions.

program—a series of instructions capable of producing a defined output from a given input.

program files—files used to store related instructions necessary to solve a particular problem (contrast with data files).

proprietory—refers to the exclusive right of a manufacturer to sell, copy, or use a program.

protected program—a program stored in computer memory that cannot be erased—as a program stored in ROM.

queue—a waiting list of processes in an MP/M system.

random access files—files that are fixed in length; the time to access any record in a random access file is the same (contrast with sequential files).

record—a collection of related items treated as a unit. A unit of disk storage.

RESET operation–the clearing of computer registers and the initialization of the system. This is normally done automatically on power-on and from a panel switch thereafter.

ROM (Read-Only-Memory)–computer storage in which data or programs are permanently or semi-permanently stored and cannot be erased during normal system operation.

RSP–resident system programs in an MP/M system. Includes ABORT, MPMSTAT, etc.

RS-323C–a standard type of serial interface used in computer systems.

sector–a physical storage entity on a floppy disk.

sector skew–the physical relationship between the logical sectors as seen by the CP/M and the physical sectors on the disk. Skewing permits faster reading of the disk. The skew is six sectors in CP/M 1.4, variable in later releases.

sequential files–files in which the time to access any record in the file is a function of its distance from the beginning of the file (contrast with random files).

serial interface–a peripheral connection in which information is sent to or from the computer one bit at a time over a single transmission line.

software–computer resources with value because of their information content (contrast with hardware and firmware); programs stored on tape, disk, or other storage media.

source disk–the originating or master disk.

source program–a program in a language other than the machine language required by the computer. A source program must be converted before it can be executed.

spooler–a program that permits a user to continue with another program while printing a file.

system–a collection of men or machines organized to accomplish a specific function.

terminal–a device capable of sending and receiving information over a channel.

track–one concentric storage path on a disk. A collection of sectors.

transient program area (TPA)–that part of the computer memory that is not directly used by the CP/M. It extends from 100H to the beginning of CP/M.

utility–a program that has a general purpose for a wide base of users, such as a file manipulation program.

variant portion–that part of the CP/M that varies with hardware. The BIOS module.

XDOS–extended disk operating system for MP/M.

XIOS–variant or hardware-dependent portion of an MP/M system.

CP/M

Bibliography

Ballinger, Charles K. "A User Compares: HDOS or CP/M . . . Which Way to Go?" *Interface Age.* Vol. 5, No. 9 (September 1980), pp. 88-91.

Barbier, Ken. "Start/Exit Routine For CP/M," *Kilobaud Microcomputing.* (October 1980), pp. 82-83.

Barbier, Ken. "Updating CP/M's STAT Utility," *Microcomputing.* Vol. 5, No. 10 (October 1981), pp. 150-154.

Bove, Tony. "The Illustrated CP/M Tutorial," *Datacast.* Vol. 1, No. 1, pp. 29-38.

Bove, Tony. "The Impatient User's Guide to CP/M," *Datacast.* Vol. 1, No. 1, pp.39-55.

Brigham, Bruce, ed. *CP/M Summary Guide.* Glastonburg, Connecticut: Rainbow Associates, 1980.

Cole, Maggie. "Gary Kildall and the Digital Research Success Story," *Infoworld.* Vol. 3, No. 10 (May 25, 1981), pp. .

Cotton, Gene. "How to Recover Erased CP/M Files," *Interface Age.* Vol. 8, No. 12 (December 1981), pp. 96-98 and 146-150.

Cotton, Gene. "How to Solve Your Damaged Disk Dilemma," *Interface Age.* Vol. 5, No. 9 (September 1980), pp. 80-86ff.

"CP/M," *80-US Journal.* (May-June 1981), pp. 52-54.

Dettmann, T. R. "CP/M For the TRS-80," *80-US Journal.* (January-February 1980), pp. 76-77.

Dwyer, Ed, Michel Lafortune, and Michel Bertrand. "Bootstrap Lets Multibus Tap CP/M-based Software," *Electronic Design.* (May 10, 1980), pp. 219-222.

Fernandez, Judi and Ruth Ashley. *Using CP/M.* New York: John Wiley Sons, 1980.

Fritzon, Richard. "The New CP/M: Is It Worth It?" *Kilobaud Microcomputing.* (July 1980), p. 66.

Haanstra, Bruce. "Optional Printing With CP/M and Microsoft Basic," *Interface Age.* Vol. 5, No. 11 (November 1980), pp. 84-86.

Hogan, Thom. *Osborne CP/M User Guide.* Berkeley: McGraw-Hill, 1981.

Kildall, Gary. "CP/M: A Family of 8- and 16-Bit Operating Systems," *Byte.* Vol. 6, No. 6 (June 1981), pp. 216-231.

Kildall, Gary. "The Evolution of an Industry: One Person's Viewpoint: The History of CP/M," *Dr. Dobb's Journal of Calisthenics & Orthodentia.* Vol. 5, No. 1 (January 1980), pp. 6-7.

Larson, James. "The Ins and Outs of CP/M," *Byte.* Vol. 6, No. 6 (June 1981), pp. 268-299.

Miller, Alan R. "CP/M: An 8080 Disk Operating System With Editor, Assembler, and Debugger," *Interface Age.* Vol. 3, No. 7 (July 1978), pp. 156-162.

Miller, Alan R. "CP/M: 8080/Z80 Disk Part 2: A Macro Assembler," *Interface Age.* Vol. 3, No. 12 (December 1978), pp. 130-135.

Miller, Alan R. "CP/M for the TRS-80 Model II: Lifeboat and FMG Corp. Versions," *Interface Age.* Vol. 5, No. 11 (November 1980), pp. 94-98.

Miller, Alan R. "Diagnostics Package For CP/M," *Interface Age.* Vol. 5, No. 10 (October 1980), p. 104.

Miller, Alan R. "Go...A CP/M Program to Branch Anywhere in Memory," *Interface Age.* Vol. 8, No. 12 (December, 1981), pp. 94-95 and 144-145.

Miller, Alan R. "The KLH Spooker For CP/M," *Interface Age.* Vol. 4, No. 4 (April 1979), pp. 136-138.

Mohler, Lorin S. "Cold Boot Automatic Program Load and Execute," *S-100 Microsystems.* Vol. 2, No. 1 (January/February 1981), p. 29.

Murtha, Steven M. and Michell Waite. *CP/M Primer.* Indianapolis: Howard W. Sams, 1980.

North, Steve. "CP/M Disk Operating System," *Creative Computing.* Vol. 4, No. 6 (November/December 1978), pp. 52-53.

Press, Larry. "An Index to CP/M Software and Vendors," *Datacast.* Vol. 1, No. 1 ( ), pp. 61-64.

"A Seminar For Independent CP/M Software Vendors," *Datacast.* Vol. 1, No. 1 ( ), pp. 24-28.

Sharp, Joseph C. "Digital Research Upgrades MP/M," *Infoworld.* Vol. 3, No. 10 (May 25, 1981), pp. 1, 47.

Skjellum, Anthony. "Go: A Utility Command Under CP/M," *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia.* Vol. 5, No. 1 (January 1980), pp. 16-17.

Stewart, John. "CP/M Primer," *Kilobaud Microcomputing.* (April 1978), pp. 30-34.

Terry, Chris. "The CP/M Connection: Part 1—Interfacing to the Operating System: Relocating CP/M," *S-100 Microsystems.* Vol. 1, No. 4 (July/August 1980), pp. 32-35.

Terry, Chris. "The CP/M Connection: Part III—Implementing the IOBYTE Function," *S-100 Microsystems.* Vol. 1, No. 6 (November/December 1980), pp. 22-26.

Terry, Chris. "The CP/M Connection: Part IV—How to Use CP/M Facilities in Your Own Application Program," *S-100 Microsystems.* Vol. 2, No. 2 (March/April 1981), pp. 18-20.

Terry, Chris. "The CP/M Connection: Part V—A Real Application," *S-100 Microsystems.* Vol. 2, No. 3 (May/June 1981), pp. 46-52.

Townsend, Carl. *CP/M Database Management.* Beaverton: dilithium Press, 1983.

Townsend, Carl. *How to Get Started With CP/M.* Beaverton: dilithium Press, 1981.

"Upgraded CP/M Floppy Disk System Now Available," *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia.* Vol. 1, No. 10 (November/December 1976), p. 51.

Vaughn, William R. "Getting Compatible With CP/M," *Mini-Micro Systems.* Vol. 14, No. 8 (August 1981), pp. 173-179.

Warner, Jim. "CP/M and CDOS Program Compatability," *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia.* Vol. 5, No. 1 (January 1980), pp. 34-36.

Warren, Jim. "First Word On a Floppy Disk Operating System," *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia.* Vol. 1, No. 4 (April 1976), p. 5.

Warren, Jim. "The Time For Floppy's is Just About Now," *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia.* Vol. 1, No. 7 (August 1976), pp. 5-6.

Zaks, Rodney. *The CP/M Handbook With MP/M.* Berkeley: Sybex, 1980.

See Also

CP/M manuals (Digital Research)

All issues of *S-100 Microcomputing*

All issues of *Lifeline* (Lifeboat Associates)

All issues of *CP/M-Net News* (3055 Waco Avenue, Simi Valley, CA 93063)

INDEX

automatic startup 106–107
bank select 110–113
BDOS 8–10, 18–19, 24, 51–53, 56–57, 122–123, 214
bias 72–77, 214
BIOS 5, 7, 8–10, 21, 24, 43, 51–57, 69, 73–81, 101, 107, 116, 122–123, 214
bootstrap 19, 214
byte 13, 16, 214
CCP 8, 10, 19, 39, 74, 107, 122, 214
data base management system 83–99, 119, 214
direct memory access 43, 109, 215
directory 4–5, 12–17, 103–104, 215
directory information 17–18, 103–104, 215
editor 126
extent 12–16, 215
FDOS 8–10, 122, 215
file 11, 12–16, 215
file control block 17–18, 58–59, 61, 115
file transfers 33–38, 215
firmware 78, 215
FORTRAN 38, 86, 215
group 12–16, 216
hardware 43, 45, 62, 109, 115, 216
indexed sequential access 87–89, 119, 216
I/O 21–31, 43–49, 51–65, 114–115, 216
IOBYTE 21–31, 216
linked directory 4–5, 216
logical information storage 11–12

logical driver 22–31
modem 45–46, 49
multi-user operating system 6, 109–118, 217
negative offset 72–77, 217
offset (see negative offset)
operating system 1–10, 217
parameters 36–37, 27
peripheral 21–22, 217
physical driver 22–31
physical information storage 11–12
PIP program 33–38, 127, 217
PL/M 4–6, 9, 218
primitive 51, 218
processor 45, 208
queue 115, 218
random access files 86, 218
random number generation 105–106
record 11, 218
RSP 111, 219
RS-232-C 47, 49, 219
sector 4, 12–16, 44, 219
sector skew 4, 15, 219
sequential files 86, 219
serial interface 21, 219
software driver 21
spooler 44, 115, 219
STAT program 28–29, 130–131
SUBMIT program 38–41
system generation 67–81, 106–107, 116–118
track 4, 12, 220
transient program area 8, 10, 12, 69, 73–75, 105, 112, 119, 122, 220
XDOS 111, 220
XIOS 111, 116, 118, 220

ABOUT THE AUTHOR

Author Carl Townsend is a consultant and writer in Portland, Oregon, and teaches computer science classes at George Fox College. He has written numerous articles and books on the microcomputers and has worked well over twenty years with the subject. Carl is also director of the Center for the Study of the Future and is a member of the World Future Society. He has a special interest in technology and the role of the personal computer in the future.

☐ Yes, send me your free catalog, **BRAINFOOD**, which lists over 90 microcomputer books covering software, hardware, business applications, general computer literacy and programming languages.

NAME _____

ADDRESS _____

CITY, STATE, ZIP _____

☐ Yes, send me your free catalog, **BRAINFOOD**, which lists over 90 microcomputer books covering software, hardware, business applications, general computer literacy and programming languages.

NAME _____

ADDRESS _____

CITY, STATE, ZIP _____

☐ Yes, send me your free catalog, **BRAINFOOD**, which lists over 90 microcomputer books covering software, hardware, business applications, general computer literacy and programming languages.

NAME _____

ADDRESS _____

CITY, STATE, ZIP _____

FREE CATALOG FREE CATALOG FREE CATA LOG FREE CATALOG FREE CATALOG FREE CATALOG FREE CATA LOG FREE CATALOG FREE CATALOG FREE CATALOG FREE CATA LOG FREE CATALOG FREE CATALOG FREE CATALOG FREE CATA LOG FREE CATALOG FREE CATALOG FREE CATALOG FREE CATA LOG FREE CATALOG FREE CATALOG FREE CATALOG FREE CATA LOG FREE CATALOG

Mail To:

dilithium Press
P.O. Box E
Beaverton, OR 97075

Or Call:

800-547-1842

FREE CATALOG FREE CATALOG FREE CATA LOG FREE CATALOG

This guide is intended for the CP/M user who has had some experience and for the serious CP/M user who wants to learn more about the operating system.

Full of information, tricks, and tutorials, *Practical Guide to CP/M* deciphers the codes of CP/M and teaches you how to:

- Interface hardware and software
- Save your programs
- Use PIP and SUBMIT
- Create CP/M systems
- Recover lost programs and more

Author Carl Townsend includes an extensive appendix to make this book an invaluable working manual for any CP/M user.

Second in dilithium Press' series on CP/M. Other books in series:

How to Get Started with CP/M
Townsend
CP/M Database Management Systems
Townsend

ISBN 0-88056-077-0

>>$14.95